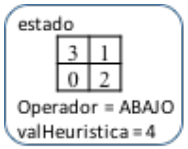
Caso 1: C está vacía

nuevo=



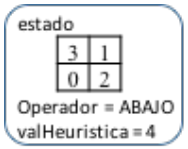
C= NULL

L=funcion(C, nuevo)

Caso 1: C está vacía

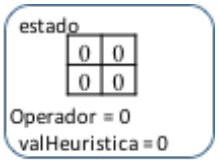
```
// Entrada:
// LISTA C:  C es una lista vacía u ordenada de nodos
// tNodo *nuevo: es un nodo de búsqueda
// Devuelve una Lista de elementos

LISTA funcion(LISTA C, tNodo *nuevo){

 //inicializaciones de las variables temporales
 // R es una lista vacía al inicio y será la lista resultante del siguiente proceso
 // nc es un tNodo que ira guardando nodos de la lista C


 if (esVacia(C))
      InsertarUltimo(el nodo nuevo en la lista R);
```

nuevo=



C= NULL

nc=

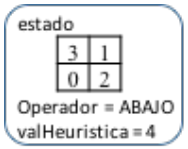


R=NULL

Caso 1: C está vacía

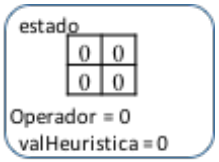
```
// Entrada:
// LISTA C:  C es una lista vacía u ordenada de nodos
// tNodo *nuevo: es un nodo de búsqueda
// Devuelve una Lista de elementos

LISTA funcion(LISTA C, tNodo *nuevo){

//inicializaciones de las variables temporales
// R es una lista vacía al inicio y será la lista resultante del siguiente proceso
// nc es un tNodo que ira guardando nodos de la lista C

 if (esVacia(C))
      InsertarUltimo(el nodo nuevo en la lista R);

  else{
            ExtraerPrimero(C,nc, . . . .);
            while (C no esté vacía y nc->valHeuristica  < nuevo->valHeuristica)
                        InsertarUltimo( en R el nodo nc)
                        C=C->next;
                        if (!(esVacia(C)))
                                    ExtraerPrimero(C,nc, . . . .));
           }//while
           InsertarUltimo( en R el nodo nuevo)
           Concatenar a R el resto de elementos de C
    }//else

 return R;
}
```

nuevo=

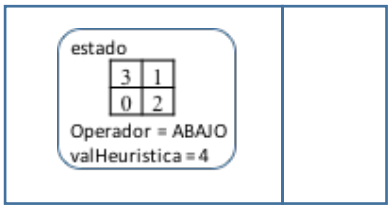


C= NULL

nc=

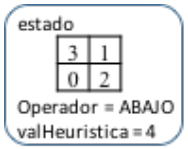


R=



Caso 1: C está vacía

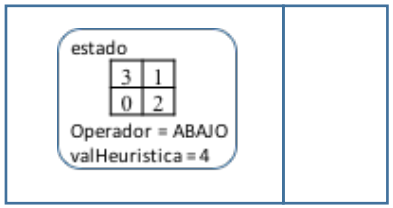
nuevo=



C= NULL

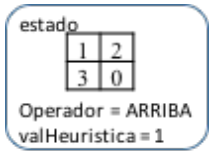
L=funcion(C, nuevo)

L=

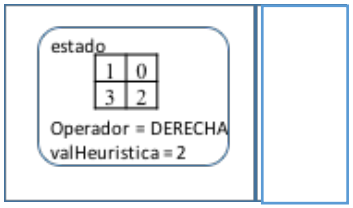


Caso 2: C contiene un elemento con h mayor

nuevo=



C=

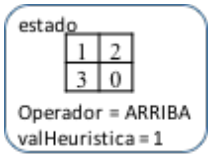


L=funcion(C, nuevo)

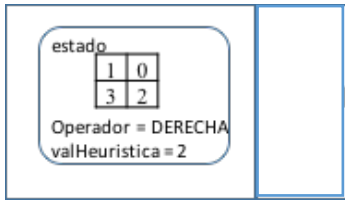
Caso 2: C contiene un elemento con h mayor

```
// Entrada:
// LISTA C:  C es una lista vacía u ordenada de nodos
// tNodo *nuevo: es un nodo de búsqueda
// Devuelve una Lista de elementos

LISTA funcion(LISTA C, tNodo *nuevo){

//inicializaciones de las variables temporales
// R es una lista vacía al inicio y será la lista resultante del siguiente proceso
// nc es un tNodo que ira guardando nodos de la lista C

if (esVacia(C))
    InsertarUltimo(el nodo nuevo en la lista R);
```

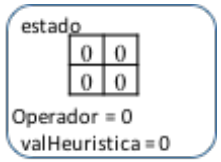
nuevo=



C=



nc=



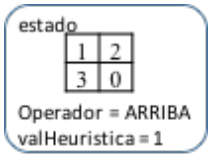
R=NULL

Caso 2: C contiene un elemento con h mayor

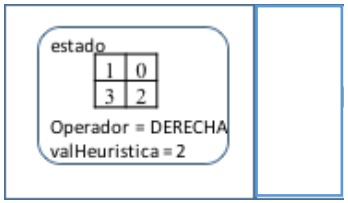
```
// Entrada:
// LISTA C:  C es una lista vacía u ordenada de nodos
// tNodo *nuevo: es un nodo de búsqueda
// Devuelve una Lista de elementos

LISTA funcion(LISTA C, tNodo *nuevo){

//inicializaciones de las variables temporales
// R es una lista vacía al inicio y será la lista resultante del siguiente proceso
// nc es un tNodo que ira guardando nodos de la lista C


 if (esVacia(C))
     InsertarUltimo(el nodo nuevo en la lista R);
else{
            ExtraerPrimero(C,nc, ....);
            while (C no esté vacía y nc->valHeuristica  < nuevo->valHeuristica)
                        InsertarUltimo( en R el nodo nc)
                        C=C->next;
                        if (!(esVacia(C)))
                                    ExtraerPrimero(C,nc, ....));
          }//while
          InsertarUltimo( en R el nodo nuevo)
          Concatenar a R el resto de elementos de C
  }//else
```

FALSO

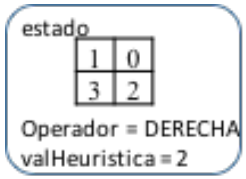
nuevo=



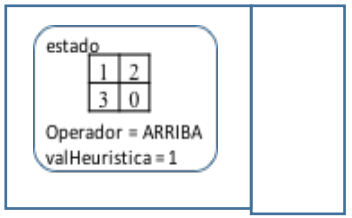
C=



nc=



R=

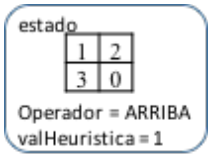


Caso 2: C contiene un elemento con h mayor

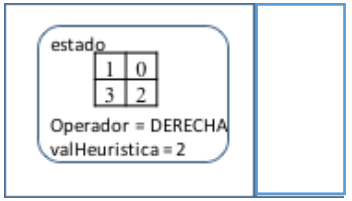
```
// Entrada:
// LISTA C:  C es una lista vacía u ordenada de nodos
// tNodo *nuevo: es un nodo de búsqueda
// Devuelve una Lista de elementos

LISTA funcion(LISTA C, tNodo *nuevo){

//inicializaciones de las variables temporales
// R es una lista vacía al inicio y será la lista resultante del siguiente proceso
// nc es un tNodo que ira guardando nodos de la lista C


 if (esVacia(C))
     InsertarUltimo(el nodo nuevo en la lista R);
 else{
          ExtraerPrimero(C,nc, ....);
          while (C no esté vacía y nc->valHeuristica  < nuevo->valHeuristica)
                    InsertarUltimo( en R el nodo nc)
                    C=C->next;
                    if (!(esVacia(C)))
                              ExtraerPrimero(C,nc, ....));
          }//while
          InsertarUltimo( en R el nodo nuevo)
          Concatenar a R el resto de elementos de C
   }//else
return R;
}
```

FALSO

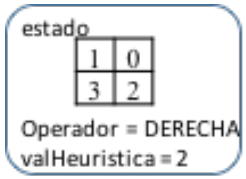
nuevo=



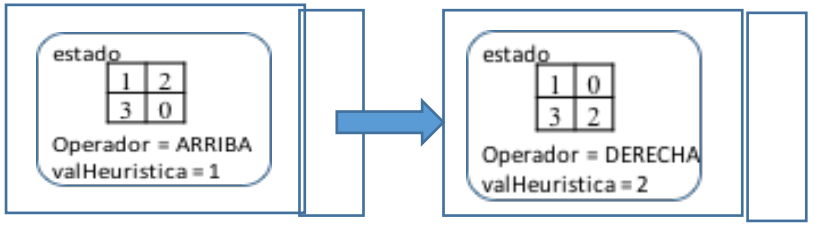
C=



nc=

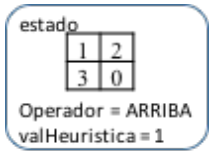


R=

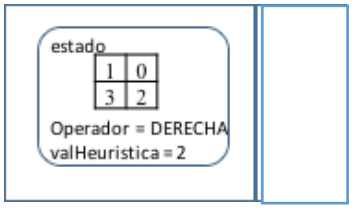


Caso 2: C contiene un elemento con h mayor

nuevo=

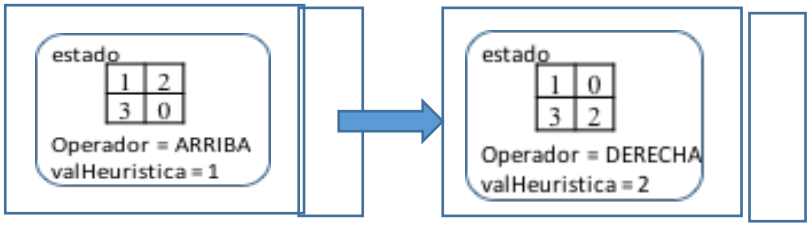


C=



L=funcion(C, nuevo)

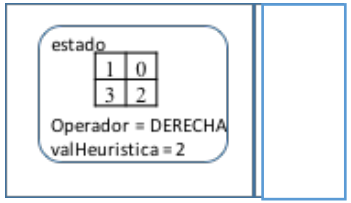
L=



Caso 3: C contiene un elemento con h menor

nuevo=

C=

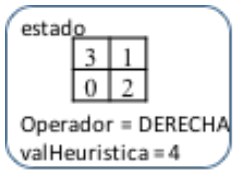


L=funcion(C, nuevo)

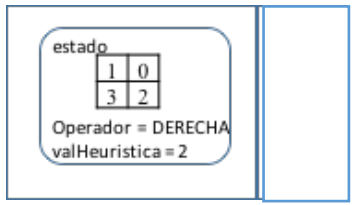
Caso 3: C contiene un elemento con h menor

```
// Entrada:
// LISTA C:  C es una lista vacía u ordenada de nodos
// tNodo *nuevo: es un nodo de búsqueda
// Devuelve una Lista de elementos

LISTA funcion(LISTA C, tNodo *nuevo){

//inicializaciones de las variables temporales
// R es una lista vacía al inicio y será la lista resultante del siguiente proceso
// nc es un tNodo que ira guardando nodos de la lista C

if (esVacia(C))
    InsertarUltimo(el nodo nuevo en la lista R);
```

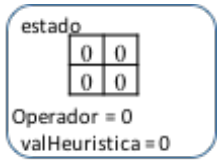
nuevo=



C=



nc=

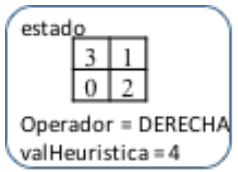


R=NULL

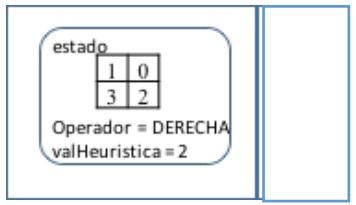
Caso 3: C contiene un elemento con h menor

```
// Entrada:
// LISTA C:  C es una lista vacía u ordenada de nodos
// tNodo *nuevo: es un nodo de búsqueda
// Devuelve una Lista de elementos

LISTA funcion(LISTA C, tNodo *nuevo){

//inicializaciones de las variables temporales
// R es una lista vacía al inicio y será la lista resultante del siguiente proceso
// nc es un tNodo que ira guardando nodos de la lista C


 if (esVacia(C))
     InsertarUltimo(el nodo nuevo en la lista R);
else{
          ExtraerPrimero(C,nc, ....);
          while (C no esté vacía y nc->valHeuristica  < nuevo->valHeuristica)
                    InsertarUltimo( en R el nodo nc)
                    C=C->next;
                    if (!(esVacia(C)))
                              ExtraerPrimero(C,nc, ....));
          }//while
          InsertarUltimo( en R el nodo nuevo)
          Concatenar a R el resto de elementos de C
  }//else
```

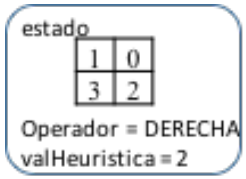
nuevo=



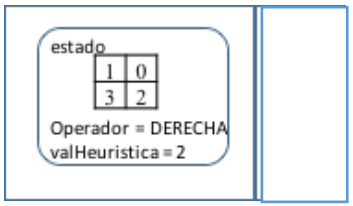
C=



nc=



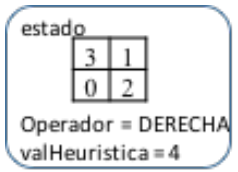
R=



Caso 3: C contiene un elemento con h menor

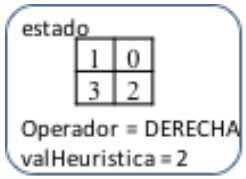
```
// Entrada:
// LISTA C:  C es una lista vacía u ordenada de nodos
// tNodo *nuevo: es un nodo de búsqueda
// Devuelve una Lista de elementos

LISTA funcion(LISTA C, tNodo *nuevo){

//inicializaciones de las variables temporales
// R es una lista vacía al inicio y será la lista resultante del siguiente proceso
// nc es un tNodo que ira guardando nodos de la lista C


 if (esVacia(C))
     InsertarUltimo(el nodo nuevo en la lista R);
else{
          ExtraerPrimero(C,nc, ....);
          while (C no esté vacía y nc->valHeuristica  < nuevo->valHeuristica)
                    InsertarUltimo( en R el nodo nc)
                    C=C->next;
                    if (!(esVacia(C)))
                              ExtraerPrimero(C,nc, ....));
          }//while
          InsertarUltimo( en R el nodo nuevo)
          Concatenar a R el resto de elementos de C
  }//else
```

nuevo=

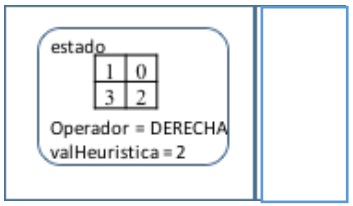


C= NULL

nc=



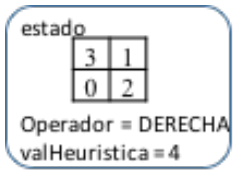
R=



Caso 3: C contiene un elemento con h menor

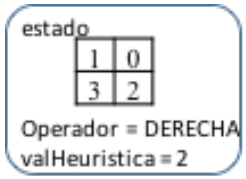
```
// Entrada:
// LISTA C:  C es una lista vacía u ordenada de nodos
// tNodo *nuevo: es un nodo de búsqueda
// Devuelve una Lista de elementos

LISTA funcion(LISTA C, tNodo *nuevo){

//inicializaciones de las variables temporales
// R es una lista vacía al inicio y será la lista resultante del siguiente proceso
// nc es un tNodo que ira guardando nodos de la lista C


 if (esVacia(C))
      InsertarUltimo(el nodo nuevo en la lista R);
 else{
            ExtraerPrimero(C,nc, ....);
            while (C no esté vacía y nc->valHeuristica  < nuevo->valHeuristica)
                        InsertarUltimo( en R el nodo nc)
                        C=C->next;
                        if (!(esVacia(C)))
                                    ExtraerPrimero(C,nc, ....));
           }//while
            InsertarUltimo( en R el nodo nuevo)
            Concatenar a R el resto de elementos de C
   }//else
return R;
```

nuevo=

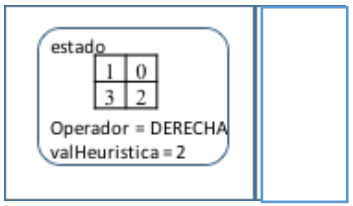


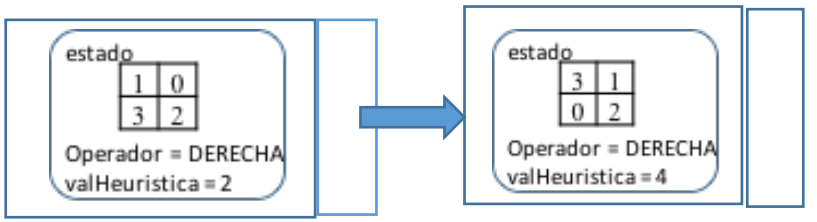
C= NULL

nc=



R=

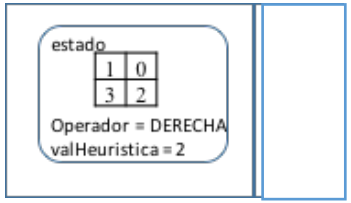




Caso 3: C contiene un elemento con h menor

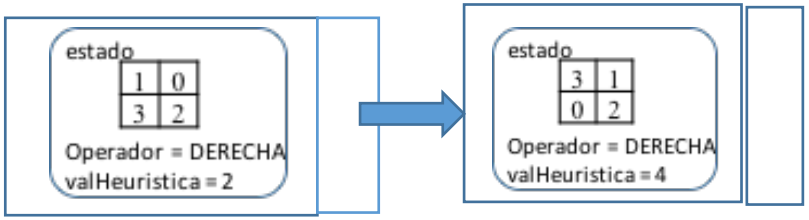
nuevo=

C=

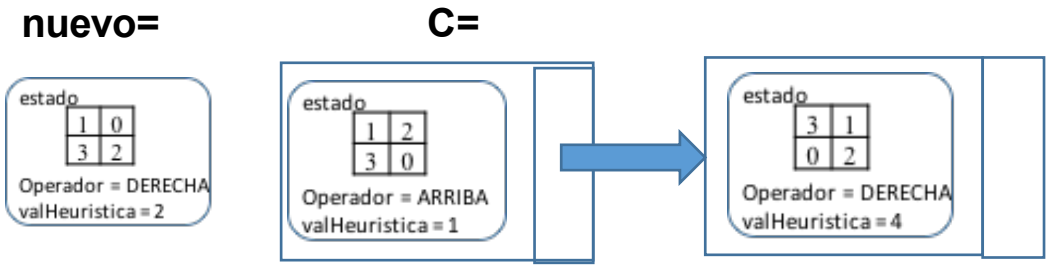


L=funcion(C, nuevo)

L=

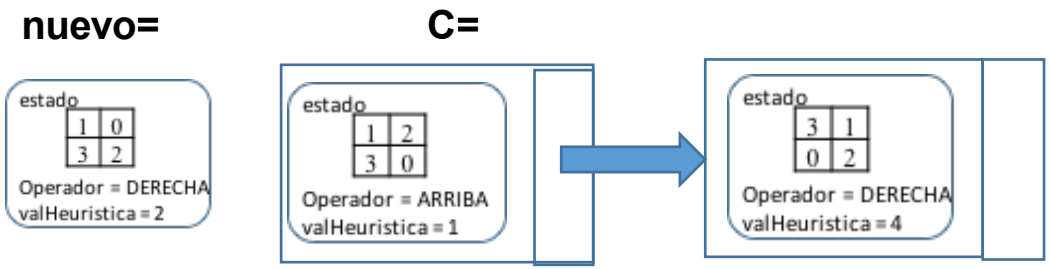


Caso 4: C contiene un elemento con h menor

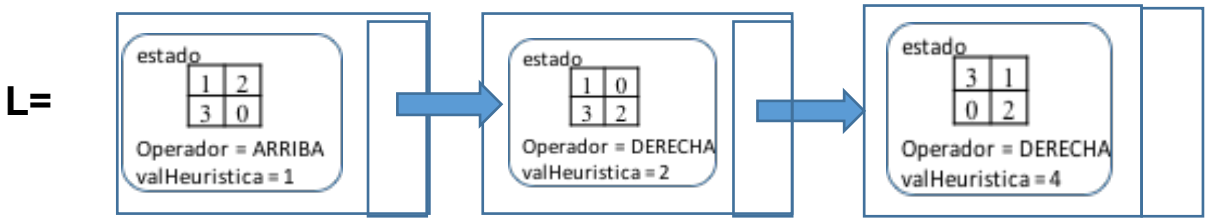


L=funcion(C, nuevo)

Caso 4: C contiene un elemento con h menor



L=funcion(C, nuevo)



Búsqueda Informada

```
InsertarPrimero(&Abiertos,(tNodo*) Inicial,sizeof(tNodo));
 while (!esVacia(Abiertos) && !objetivo){
        Actual=(tNodo*) calloc(1,sizeof(tNodo));
        ExtraerPrimero(Abiertos,Actual, sizeof(tNodo));
        EliminarPrimero(&Abiertos);
        objetivo=testObjetivo(Actual->estado);
        if (!objetivo){
                repe=buscaRepeHeu(Actual,Cerrados,tipo);
                if (!repe){
                        Sucesores = expandir(Actual);
                        Abiertos=ordenarLista(Abiertos,Sucesores);
                        InsertarPrimero(&Cerrados,(tNodo*) Actual,sizeof(tNodo));
                }!repe
        }!objetivo
 }//while
```

```
//LISTA ordenarLista(LISTA A, LISTA Suc)
//A es una lista ordenada o vacía
//Suc es una lista de nodos en cualquier orden
//Devuelve una lista ordenada conteniendo todos los elementos de A y Suc
LISTA ordenarLista(LISTA A, LISTA Suc){
   /*Insercion ordenada de nodos sucesores en la lista ordenada A */
  Bucle:
            Extraer nodos de Suc, de uno en uno
            Insertar un nodo en la lista ordenada A
  FinBucle
 return(A); //Devuelve la lista ordenada
}//ordenarLista
```