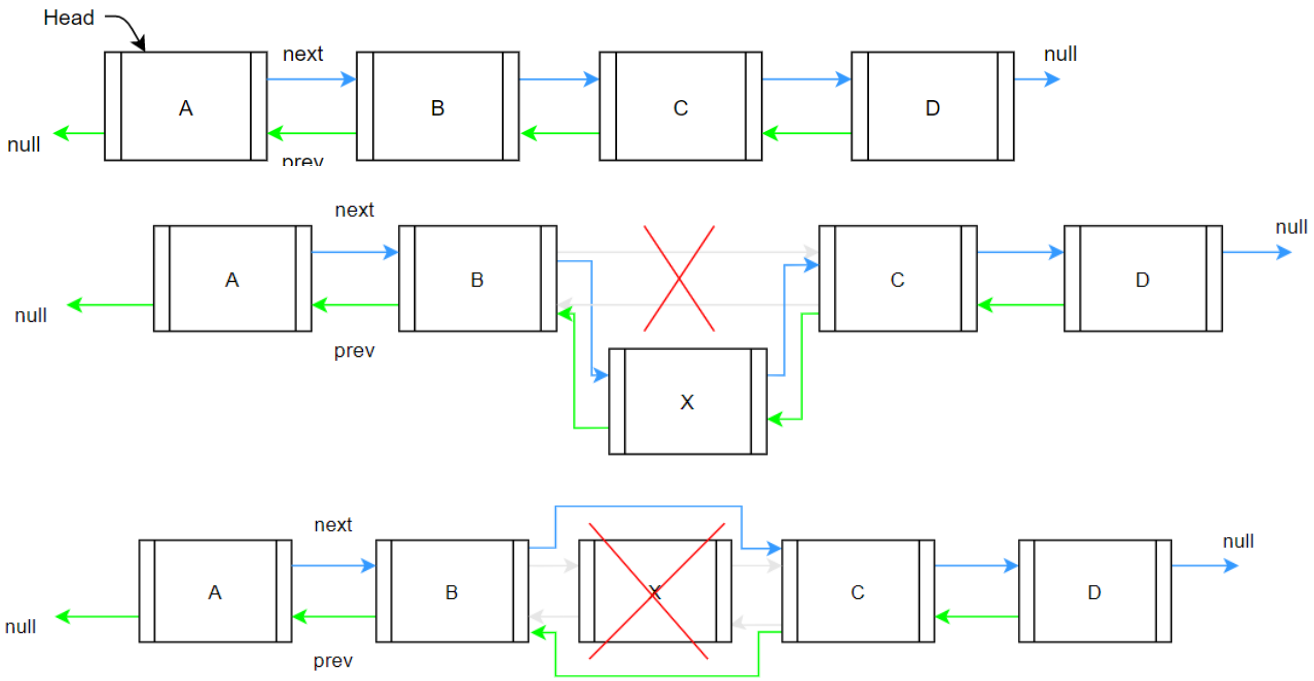
[A03] Doppelt Verkettete Liste

Grafische Beschreibung



Textuelle Beschreibung

Jeder Knoten einer doppelt verketteten Liste enthält neben seinen gespeicherten Daten auch noch zwei Links. Der previous Link zeigt seinen Vorgänger, außer es handelt sich um den ersten Knoten, dann zeigt dieser auf null. Der next Link verweist auf den nachfolgenden Knoten, nur beim Letzten der Liste zeigt dieser auf null. Dadurch, dass zwei Zeiger existieren, kann die doppelt verkettete Liste in beide Richtungen durchlaufen werden. Der zweite Zeiger sorgt zwar für erhöhten Speicherbedarf, erleichtert aber das Einfügen bzw. Löschen eines Knotens, weil die Knoten davor und danach bekannt sind.

Beim Einfügen eines neuen Knotens wird überprüft, ob es bereits einen ersten Knoten gibt. Existiert keiner, so wird der neue Knoten als erstes und letztes Element zugewiesen. Wenn ein Knoten in der Liste existiert, wird der letzte Knoten in der Liste zwischengespeichert. Der neue Knoten erhält den zwischengespeicherten Knoten als Vorgänger und er wird als dessen Nachfolger gesetzt. Im letzten Schritt setzt man den neuen Knoten als neues letztes Element der Liste.

Soll der neue Knoten nach einem bestimmten Element eingefügt werden, muss dieses einmal existieren. Danach benötigt man den aktuellen Knoten und dessen Nachfolger, vorausgesetzt es handelt sich nicht um den letzten Knoten in der Liste. Der einzufügende Knoten wird der neue Nachfolger des aktuellen Knotens und der neue Vorgänger des alten Nachfolgers des aktuellen Knotens. Anschließend werden der aktuelle Knoten und dessen ursprünglicher Nachfolger zum Vorgänger bzw. Nachfolger des neu eingefügten Knotens.

Wird der aktuelle Knoten gelöscht und dieser ist weder der erste noch der letzte Knoten in der Liste, dann wird der Vorgänger des zu löschenden Knotens der neue Vorgänger von dessen Nachfolger und umgekehrt. Der Nachfolger des zu löschenden Knotens wird zum neuen Nachfolger von dessen Vorgänger. Anschließend wird der Zeiger auf das nächste Element gesetzt. Löscht man den ersten

Knoten wird dessen Nachfolger der neue erste Knoten und wenn man den letzten Knoten löscht, wird dessen Vorgänger zum neuen letzten Knoten.

Wenn man nicht weiß welcher Knoten gelöscht wird, wird grundsätzlich das gleiche Prinzip angewendet, wie beim Löschen des aktuellen Knotens mit dem Unterschied, dass davor noch in einer Schleife überprüft werden muss, ob der aktuelle Knoten dem zu löschenden Knoten entspricht.

Wichtige Elemente

```
/** Big O notation -> O(1) */
public void add(T element) {

   Node<T> node = new Node<>(element);    // create new node element

   // Initialize first element
   if (first == null){          // check if new element is first element in list
      first = node;             // assign new node as first
      last = node;              // assign new node as last
   } else {
      Node<T> currentLastNode = last;       // assign current last node to variable
      currentLastNode.setNext(node);        // set new node as next node for the previous first
      node.setPrevious(currentLastNode);  // set current first node as previous
      last = node;                          // set new node as last
   }
   nodeCount++;                    // increase node counter
}

/** Big O notation -> O(1) */
public void removeCurrent() throws CurrentNotSetException {

   if (current == null) throw new CurrentNotSetException();

   if (current.getPrevious() != null)  // do not try to set next on a null
      current.getPrevious().setNext(current.getNext()); // set from the previous element the new next which is
the next element of the one which should be removed
   else // first found -> if (currentNode.equals(first)){}
      first = current.getNext(); // current should be removed therefore "next of current" is new first if current
equals first

   if (current.getNext() != null) {    // do not try to set previous on a null
      current.getNext().setPrevious(current.getPrevious());  // set from the next element the new previous which
is the previous element of the one which should be removed
      current = current.getNext(); //current pointer handling
   } else { // last found -> if (currentNode.equals(last)){}
      last = current.getPrevious();
      current = current.getPrevious(); //current pointer handling
   }
   nodeCount--;
}
```

```java
/** Big O notation -> O(1) */
public void insertAfterCurrentAndMove(T element) throws CurrentNotSetException { //TODO - refactor

    if (current == null)
        throw new CurrentNotSetException();

    Node<T> insertNode = new Node<>(element);
    Node<T> tmpNext = current.getNext();
    Node<T> tmpCurrent = current;

    tmpCurrent.setNext(insertNode);

    if (tmpNext != null) // not possible to assign if "next" is null
        tmpNext.setPrevious(insertNode);

    insertNode.setPrevious(tmpCurrent);
    insertNode.setNext(tmpNext);

    current = insertNode;

    nodeCount++;
}
```