שעור 12

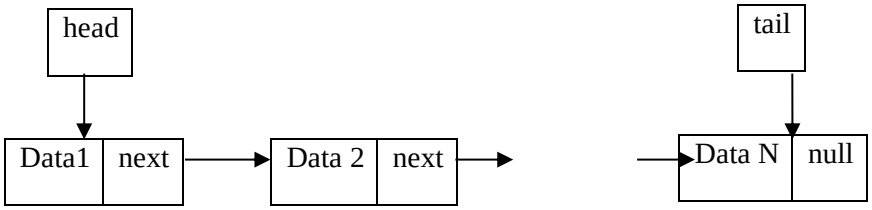
רשימות מקושרות – Linked Lists

מערכים:

על מנת לאחסן מספר משתנים מסוג טיפוס כלשהו, אין צורך ליצור מספר משתנים, אלא ניתן לבנות מערך של N איברים מאותו סוג. חסרונו של המערך הוא, שהוא מוחזק בזיכרון ברצף. מכוון שבדרך כלל ניצור מערך ממוין, ניאלץ להזיז את כל האיברים במערך בעת הוספת אובייקט למערך או מחיקתו ממנו. פעולה זו בזבזנית מאוד.

רשימה מקושרת

רשימה מקושרת היא רצף לוגי של משתנים מסוג מחלקה. הרצף הוא אינו רצף פיזי בזיכרון, כל אובייקט הרשימה מכיל מצביע לאובייקט הבא:



הרשימה מצביע על האובייקט הראשון, האובייקט הראשון מצביע על האובייקט השני וכך עד לאחרון, האובייקט האחרון מצביע על null.

רשימה מקושרת אינה מסודרת ברצף בזיכרון כמו מערך.

לכל רשימה יש "ראש רשימה" (head) , שהוא אובייקט, בהתחלה ה- head מאותחל ב-null (רשימה ריקה), וכשמוסיפים אובייקטים הוא מצביע לאובייקט ראשון ברשימה.

הוספת אובייקט לרשימה מקושרת:

על מנת להוסיף אובייקט לרשימה קודם צריך לבנות אובייקט חדש.
כדי להוסיף אובייקט חדש לסוף הרשימה צריך לשנות את המצביע של האובייקט האחרון כך שיצביע לאובייקט החדש.
כדי להוסיף אובייקט חדש באופן ממוין צריך לשנות את המצביע של האובייקט, שנמצא מקום אחד קודם לפני האובייקט החדש, כך שיצביע לאובייקט החדש. כמו כן המצביע של האובייקט החדש יצביע עתה אל האובייקט שאחריו.
על מנת לחסוך זמן הנדרש להוספת אובייקט לרשימה מקושרת כדאי לשמור גם את האיבר הארון של הרשימה הנקרא tail.
יש להתחשב במקרה קצה שבו יש להוסיף ישירות ל-head.

מחיקת אובייקט מרשימה מקושרת:

על מנת למחוק אובייקט, יש לנתק את המצביע אליו, כך שיצביע על האובייקט שאחריו.
יש להתחשב במקרה קצה שבו יש למחוק את האובייקט האחרון. כמו כן יש להתחשב במקרה שבו יש למחוק את האובייקט הראשון.

מעבר על מרשימה מקושרת:

כאשר רוצים לרוץ על כל האובייקטים של (למשל להדפסה או לחיפוש) יש לרוץ עליהם באמצעות מצביע עזר החל מה-head. תנאי העצירה יהי כאשר המצביע האחרון יגיע ל-null.

יתרונות של רשימה מקושרת:

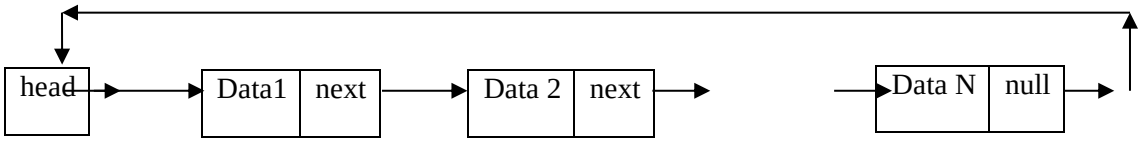
1. אפשר להקצות באופן דינאמי מספר רב של אובייקטים.
2. הוספת אובייקט ומחיקת אובייקט נעשים ישירות ולכן אין צורך להזיז את שאר האובייקטים.
3. האובייקטים אינם מסודרים ברצף, לכן הקצאת הזיכרון יותר בטוחה.

חסרונות של רשימה מקושרת:

1. האובייקטים אינם מסודרים ברצף, ולכן המעבר עליהם איטי יותר.
2. ברשימה מקושרת הגישה לאיבר אינה ישירה ואילו במערך היא ישירה.

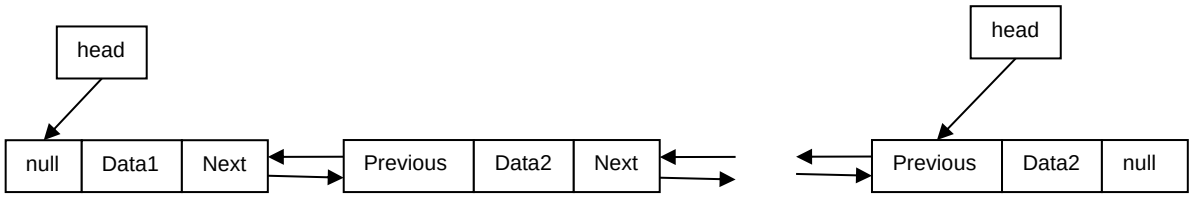
רשימות מקושרות מעגלית

רשימה מקושרת מעגלית היא רשימה רגילה שהאיבר האחרון שלה אנו מצביע ל-null אלא אל האיבר הראשון.
גם לרשימה זו קיימים head ו-tail , ולכן בכל הלולאות נרוץ על הרשימה באמצעות מצביע עזר המתחיל מ-head , ותנאי העצירה הוא כל עוד לא חזרנו ל-head.



רשימה מקושרת דו-כיוונית

ברשימה מקושרת דו-כיוונית לכל איבר יש שני מצביעים: מצביע לאיבר הבא ומצביע לאיבר קודם. במקרה זה המצביע לאיבר קודם של head הוא null, הצביעה לאיבר הבא של איבר אחרון הוא גם null :



קל יותר לעבוד עם רשימה דו-כיוונית, מפני שאפשר לנווט בה קדימה ואחורה ברשימת האובייקטים.

pseudo-code של רשימה מקושרת חד-כיוונית

```
Class Node
   Object data,
   Node next
// constructor
   Create New Node (newData, nextNode)
       data = newData
       next = nextNode
   End- Create-New-Node
End-Class-Node

Linked List Initialization()
      Node head = null
      Node tail = null
      size = 0;
End-Linked-List-Initialization()

Add(newData)
      if (head = null) then
         head = tail = Create New Node(newData, null)
      else
         newNode = Create New Node (newData, null)
         next[tail] = newNode
         tail = newNode
      end-if
      size = size + 1;
End-Add
```

```
Remove(x)
      Node ans = null
      // the empty list
      if (head = null) then
            ans = null
      // the node to remove is the first node (head)
      else if (data[head] = x) then
            ans = data[head]
            head = next[head]
            size = size - 1;
      else
            // search node to remove
            node = prev = head
            while((next[node] ≠ null) AND (data[node] ≠ x))
                  prev = node
                  node = next[node]
            end-while
            // the data was found, remove (not last node)
            if (node.next ≠ null) then
                  next[prev] = next[node]
                  ans = data[node]
                  size = size - 1;
            // the node to remove is the last node (tail)
            else if (node.next = null && data[node] = x) then
                  ans = data[tail]
                  next[prev] = null
                  tail = prev
                  size = size - 1;
            end-if
      end-if
      return ans
end-remove
```

pseudo-code של רשימה מקושרת דו-כיוונית

```
Class Node
   Object data,
   Node next, prev
// constructor
   Create New Node (newData, prevNode, nextNode)
       data = newData
       next = newNode
       prev = prevNode
   End-Node
End-Class-Node

Double Linked List Initialization()
      Node head = null
      Node tail = null
      size = 0;
End-Linked-List-Initialization()

Add(newData)
      if (head = null) then
         head = tail = Create New Node(newData, null, null)
      else
         newNode = Create New Node (newData, tail, null)
         next[tail] = newNode
         tail = newNode
      end-if
      size = size + 1;
End-Add

RemoveFirst()
      ans = null
      if (head ≠ null) then
            ans = data[head]
            head = next[head]
            if (head ≠ null) then
                  prev[head] = null
            end-if
            size = size - 1
      end-if
      return ans
end RemoveFirst
```

RemoveLast()

```
      ans = null
      if (tail ≠ null) then
            ans = data[tail]
            tail = prev[tail]
            if (tail ≠ null) then
                  next[tail] = null
            end-if
            size = size – 1
      end-if
      return ans
end-RemoveLast
```

Search(x)

```
      node = head
      while (node ≠ null AND data[node] ≠ x)
            node = next[node]
      end-while
      return node
end-Search
```

Remove(x)

```
      node = Search(x)
      if (node ≠ null) then
            if (prev[node] = null) ans = RemoveFirst()
            else if (next[node] = null) ans = RemoveLast()
            else
                  ans = data[node]
                  next[prev[node]] = next[node]
                  prev[next[node]] = prev[node]
                  size = size – 1
            end-if
      end-if
      return ans
end-Remove
```

מחלקת LinkedList של Java

כמובן מחלקת **LinkedList** קיימת ב-java, היא נמצאת בספרייה **java.util**
(import java.util.LinkedList;)
רשימה מקושרת של java היא כתובה בצורה כזו (כמו כל מבנה נתונים אחר) שניתן להתאים אותו לכל סוגי הנתונים.
המבנה זה נקראה מבנה נתונים גנרי (Generic Type).

מהו איטרטור (Iterator)

לולאה מאפשרת לעבור על מספר איברים שמאוחסנים בזיכרון. כל מהלך שלם של לולאה מכונה איטרציה.
שלבים חיוניים בהגדרת לולאה:

1. אתחול (**Iterator<String> iter = list.iterator()**).
2. תנאי עצירה (בדיקה אם קיימים איברים נוספים **iter.hasNext()**) .
3. קידום – מעבר לאיבר הבא (**iter.next()**).
4. שלבים אלו הכרחיים למעבר על רצף איברים, שאינם תלויים בצורה שבה הם מאוחסנים בזיכרון.
   ממשק **Iterator** נמצאת בספרייה **java.util** (**import java.util.Iterator**) .
5. כמו כל מבנה נתונים רשימה מקושרת מאפשרת להגדיר איטרטור (**Iterator**):

מחלקת LinkedList של Java - דוגמה

```
import java.util.LinkedList;
import java.util.Iterator;
public class MyLinkedListJAVA {
      public static void main(String[] args) {
            LinkedList<String> list = new LinkedList<String>();
            String s="abcd";
            list.add(s);
            list.add("e");
            list.add("xyz");
            list.add("fg");
            Iterator<String> iter = list.iterator();
            while(iter.hasNext()){
                  System.out.println(iter.next());
            }
            System.out.println("s= "+list.contains(s));
            System.out.println("xyt= "+list.contains("xyt"));
            System.out.println("list(2)= "+list.get(2));
            System.out.println("first= "+list.getFirst());
            System.out.println("last= "+list.getLast());
            System.out.println("index of e= "+list.indexOf("e"));
            System.out.println("is empty= "+list.isEmpty());
            System.out.println("size = "+list.size());
            list.remove(1);
            list.remove("xyz");
            System.out.println("size = "+list.size());
            iter = list.iterator();
            while(iter.hasNext()){
                  System.out.println(iter.next());
            }
      }
}
```