**חומרים שהוכנו על-ידי**

**משתתפי קורס מורים מובילים תשע"ה**

**ניתן להשתמש בחומרים לצורך הוראה בלבד.**

**לא ניתן לפרסם את החומרים או לעשות בהם כל שימוש מסחרי**

**ללא קבלת אישור מראש מצוות הפיתוח**

**כתיבה ועריכה:**

**אוי גרינולד**

**שאלה שניתנה בבוחן בנושא חוליות, כתבה: EVI, תיכון חוגים חיפה, פברואר, 2015**

כתוב פעולה המקבלת שרשרת חוליות של שלמים . עבור כל חוליה: אם ערך החוליה שונה מאפס, הפעולה תוסיף חוליה עם הערך הנגדי , אחרת הפעולה תמחק את החוליה.

הנח: יש לפחות חוליה אחת שערכה שונה מ אפס.

לדוגמא, עבור שרשרת החוליות הבאה:

12

7-

6

0

$$chain$$

null

-33

0

0

$$chain$$

אחרי ביצוע הפעולה שלך, שרשרת החוליות תראה כך

33

33-

12-

12

7

7-

6-

6

.

null

לכאורה שאלה פשוטה. אכן, עבור רשימה כפי שהייתה מיוצגת בתכנית הישנה, הרשימה של פרק 9 הפתרון אכן מאוד פשוט הן רקורסיבי והן איטרטיבי.

שים לב כדי שיהיה אפשר לממש את הפעולה בעזרת רשימה = שרשרת חוליות , היה צורך להוסיף את ההנחה שיש לפחות איבר אחד שונה מאפס.

בפתרון של שרשרת חוליות צצו קשיים שהתלמידים לא הבחינו בהם.

**נסו לחשוב איזה טעויות חלו בפתרונות התלמידים?**

**איך הייתם מציעים לתלמידים לגשת לשאלה כזו?**

**הטעויות שהיו**:

* לא התייחסו לעובדה שיכולים להיות יותר מאפס אחד בתחילת הרשימה.
* התייחסות לקויה בקידום ההפניה לחוליה ברשימה
* לא הבחינו בין מקרה של מחיקה למקרה של הוספה
* בהוספה כשלא קדמו את ההפניה התקבלה לולאה אינסופית
* בקרה של אפס , היו שמחקו את החוליה העוקבת לזו שערכה אפס במקום את עצמה
* לא טיפלו במקרה של חוליה אחרונה

**מה ברשות התלמידים?**

התלמידים הכינו מחלקת שירות עם אוסף פעולות על שרשרת חוליות של שלמים

וביניהן הפעולות הבאות:

/\*\*

 \* deal chain of Node<Integer>

יש להתאים הפתרון לפעולות העדכניות של Node

 \* @author Evi

 \* @version 29/10/07

 \*/

import java.util.Scanner;

import unit4.collectionsLib.\*;

public class **ChainI**

{

 static Scanner in = new Scanner(System.in);

 /\*\*

 \* create chain of Node<Integer>

 \* @return first Node<Integer> of Chain

 \*/

 public static Node<Integer> **createChain**()

 {

 Integer info;

 Node<Integer> first, node, pos;

 System.out.println("Enter first value, not 99");

 info = in.nextInt();

 first = new Node<Integer>(info);

 pos = first;

 System.out.println("Enter value, 99 --> to stop");

 info = in.nextInt();

 while (info != 99)

 {

 node = new Node<Integer>(info);

 pos.setNext(node);

 pos = node;

 System.out.println("Enter value, 99 --> to stop");

 info = in.nextInt();

 } // end while

 return first;

 }

 /\*\*

 \* prints ChainI

 \* @param chain: Node<Integer> - first

 \* assumption: chain != null

 \*/

 public static void **printChain**(Node<Integer> chain)

 {

 if (chain == null)

 System.out.println("null");

 else // chain.getNext() != null

 {

 System.out.print(chain.toString()+ " --> ");

 printChain(chain.getNext());

 }

 }

 /\*\*

 \* @param chain: Node<Integer>, first

 \* @param step: integer

 \* @return Node<Integer> in chain after step steps

 \*/

 public static Node<Integer> **gotoNode**(Node<Integer> chain, int step)

 {

 if (chain==null || step==1)

 return chain;

 // chain !-=null && step>1

 return gotoNode(chain.getNext(), step-1);

 }

 /\*\*

 \* insert info to new IntNode after node

 \* @param pos: Node<Integer>

 \* @param info: Integer

 \* assumption: pos!=null and refer to a node in chain

 \*/

 public static void **insert**(Node<Integer> pos, Integer info)

 {

 Node<Integer> temp = new Node<Integer>(info,pos.getNext());

 pos.setNext(temp);

 }

 /\*\*

 \* delete Node<Integer> in position where prevNode points to

 \* @param prevNode: Node<Integer>

 \*/

 public static void **delete**(Node<Integer> prevNode)

 {

 Node<Integer> node = prevNode.getNext();

 prevNode.setNext(node.getNext());

 node.setNext(null);

 }

 /\*\*

 \* insert node with info before first

 \* @param chain: Node<Integer>, refer to first node in chain of nodes

 \* @param info : int

 \* The concept: add in 2nd position and switch values between 1st and 2nd nodes

 \*/

 public static void **insertBeforeFirst**(Node<Integer> chain,int info)

 {

 Node<Integer> node = new Node<Integer>(info, chain.getNext());

 chain.setNext(node);

 node.setInfo(chain.getInfo());

 chain.setInfo(info);

 }

 /\*\*

 \* delete first node in chain

 \* @param chain : Node<Integer>

 \* assumption: there are at least 2 nodes in chain

 \* The concept: change value in 1st node to be the value in 2nd node

 \* and deletes the 2nd node

 \*/

 public static void **deleteFirst**(Node<Integer> chain)

 {

 if (chain.getNext()!=null)

 {

 chain.setInfo(chain.getNext().getInfo());

 Node<Integer> temp = chain.getNext();

 chain.setNext(temp.getNext());

 temp.setNext(null);

 }

 }

// ------- main ------------

 public static void **main**(String[] args)

 {

 Node<Integer> myChain = ChainI.createChain();

 ChainI.printChain(myChain);

 ChainI.insertBeforeFirst(myChain, 33);

 ChainI.printChain(myChain);

 ChainI.deleteFirst(myChain);

 ChainI.printChain(myChain);

 }

}

**הצעת פתרון הכוללת פירוק הבעיה לפעולות עזר**

* מחיקת אפסים בראש הרשימה
* מחיקת כל האפסים ברשימה
* הוספת מספר נגדי אחרי כל מספר ברשימה. עכשיו הרשימה היא ללא אפסים.

**פתרון אפשרי:**

 // מוסיף נגדי לכל מספר ששונה מ 0 ומוחק אפסים

 public static void **change**(Node<Integer> chain)

 {

 Node<Integer> pos, node;

 int x;

 delZeroStart(chain);

 delZero(chain);

 addNegative(chain);

 }

 public static void **delZeroStart**(Node<Integer> chain)

 {// מוחק אפסים בתחילת הרשימה

 Node<Integer> pos;

 if (chain.getInfo()==0 && chain.getNext()!=null)

 {

 pos = chain.getNext();

 chain.setInfo(pos.getInfo());

 chain.setNext(pos.getNext());

 pos.setNext(null);

 delZeroStart(chain);

 }

 }

 public static void **delZero**(Node<Integer> chain)

 {// מוחק אפסים בתווך, לא בהצלחה ולא אחרון

 Node<Integer> pos;

 if (chain.getNext()!=null)

 {

 pos = chain.getNext();

 if (pos.getInfo() == 0)

 {// removes it

 chain.setNext(pos.getNext());

 pos.setNext(null);

 }// end if

 else

 {// pos.getInfo() != 0

 chain = chain.getNext();

 }

 delZero(chain);

 }// end if

 }

 public static void **addNegative**(Node<Integer> chain)

 {// מוסיף חוליה עם מספר נגדי אחרי כל חוליה. אין אפסים ברשימה

 Node<Integer> node;

 int x;

 if (chain!=null)

 {

 x = chain.getInfo();

 node = new Node<Integer>(-x, chain.getNext());

 chain.setNext(node);

 addNegative(node.getNext());

 }

}