

קריית החינוך השטח שנתון

פארק המדע, נס ציונה

Node<T> הגנרית

תיאור הפעולה	חתימת הפעולה
בנאים:	
פעולה הבונה חוליה שבערך value שלה יהיה x וב- next שלה יהיה הערך null	Node (T x)
פעולה הבונה חוליה שבערך value שלה יהיה x וב- next שלה יהיה הערך next (ערך המועבר כפרמטר יכול להיות גם null)	Node (T x , Node<T> next)
שאלות:	
אם T מחלקה עוטפת לטיפוס בסיסי (Integer, Double, Character) יוחזר ערך החוליה, ואם הפניה לעצם, תוחזר הפניה לעצם זה	T GetValue ()
מוחזרת הפניה לחוליה הבאה	Node<T> GetNext ()
פעולה המחזירה אמת אם next מפנה לחוליה נוספת (כלומר אינו null) ושקר אחרת	bool HasNext ()
פעולה המחזירה מחרוזת המתארת את מצב העצם (*) סיבוכיות: אם T עצם מטיפוס פשוט $O(1)$ ואם T מייצג אוסף $O(T)$ (אורך האוסף כפונקציה של n)	string ToString ()
פקודות:	
הפעולה משנה (מעדכנת) את ערך ה- value של החוליה ל- x	void SetValue (T x)
הפעולה משנה את ערכו של next להיות next חדש (ערך next המתקבל כפרמטר יכול להיות גם null)	void SetNext (Node<T> next)

```

public class Node<T>
{
    private T value;
    private Node<T> next;

    2 references
    public Node(T value) { this.value = value; this.next = null; }
    0 references
    public Node(T value, Node<T> next) { this.value = value; this.next = next; }
    1 reference
    public T GetValue() { return this.value; }
    0 references
    public void SetValue(T value) { this.value = value; }
    8 references
    public Node<T> GetNext() { return this.next; }
    5 references
    public void SetNext(Node<T> next) { this.next = next; }
    2 references
    public bool HasNext() { return (this.next != null); }
    0 references
    public override string ToString() { return this.value.ToString(); }
}

```

שם הפעולה	תיאור
CopyList (Node<string> head)	הפעולה משכפלת את הרשימה ומחזירה מצביע לרשימה חדשה.
Reverse (Node<double> head)	הפעולה מחזירה מצביע לרשימה חדשה בסדר הפוך (החוליה האחרונה היא הראשונה)
Index (Node<int> head, int value)	הפעולה מחפשת את האיבר הראשון עם הערך שהתקבל ומחזירה את המיקום שלו ברשימה.
Find (Node<string> head, string value)	הפעולה מחפשת את האיבר הראשון עם הערך שהתקבל ומחזירה את האיבר (מצביע לאיבר).
Slice (Node<char> head, int start, int end)	הפעולה מחזירה תת רשימה של הרשימה שהתקבלה מן האיבר start ועד לאיבר end. (אין צורך לבדוק תקינות האינדקסים).
Equal1(Node<int> L1, Node<int> L2)	א. הפעולה מקבלת שתי רשימות ובודקת אם הערכים שלהם זהים ומסודרים באותו סדר . אם כן, מחזירה אמת. אחרת שקר.
Equal1(Node<int> L1, Node<int> L2)	ב. הפעולה מקבלת שתי רשימות ובודקת אם הערכים שלהם זהים ולא דווקא מסודרים באותו סדר . אם כן, מחזירה אמת. אחרת שקר.
Union(Node<double> L1, Node<double> L2)	הפעולה מקבלת שתי רשימות ומחזירה רשימה מאוחדת L2 אחרי L1. אין צורך להעתיק את האיברים (שימוש באיברים הקיימים).
Intersect(Node<string> L1, Node<string> L2)	הפעולה מקבלת שתי רשימות ומחזירה רשימה שבה האיברים בעלי הערך הזהה.
Unique(Node<int> head)	הפעולה מוחקת מן הרשימה את האיברים הכפולים ומשאירה רק איבר אחד מכל ערך.
Max(Node<int> head)	הפעולה מחזירה את האיבר עם הערך הגדול ביותר. אתם יכולים להניח כי ניתן להשוות בין הערכים באמצעות <.
UnionSort(Node<double> L1, Node<double> L2)	הפעולה מקבלת שתי רשימות ממוינות (אתם יכולים להניח כי ניתן להשוות בין הערכים באמצעות < או >). הפעולה מחזירה רשימה ממוזגת וממוינת (אין להעתיק איברים).

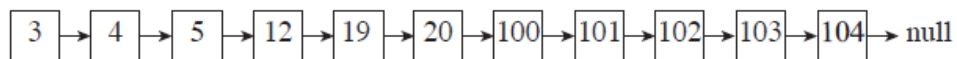
commonValue (Node<int> head)	הפעולה מקבלת רשימה של int שהערכים בין 1 ל- 100. הפעולה תחזיר את המספר השכיח ביותר (שמופיע הכי הרבה פעמים). ניתן להשתמש במבנה נתונים נוסף.
Consistent(Node<int> head)	הפעולה מקבלת רשימה ומחזירה אמת היא ערכי האיברים עולים ממש או יורדים ממש. אתם יכולים להניח כי ניתן לבצע השוואה בין האיברים אמצעות < או >.
isPalidrome(Node<char> head)	הפעולה מקבלת רשימה ומחזירה true אם היא פולינדרום. רמז: ניתן להשתמש בשתי פעולות קודמות שכתבתם והתשובה מסכמת בשלוש שורות.
Contains(Node<int> L1, Node<int> L2)	הפעולה מקבלת שתי רשימות ומחזירה אמת אם רשימה L2 מוכלת ב L1. $L2 \subseteq L1$
Zipper (Node<string> L1, Node<string> L2)	הפעולה מקבלת שתי רשימות ומאחדת אותן לרשימה אחת בצורת ריצ'רצ (איבר מ L1, איבר מ L2 וחוזר חלילה).
Positive(Node<double>)	פעולה המקבלת רשימה של מספרים ממשיים, יוצרת רשימה של המספרים החיוביים מתוך הרשימה, מדפיסה (באמצעות PrintList()) את הרשימה החדשה ולאחריה את הרשימה המקורית, ומחזירה את הרשימה החדשה.
SumOfsubList(Node<int> L)	הפעולה מקבלת שרשרת חוליות של מספרים שלמים ומחזירה שרשרת חדשה כך: עבור כל תת-רשימה של מספרים עולים ב L, שבה כל מספר גדול ממש מקודמו, יופיע ברשימה החדשה איבר אחד עם סכום תת הסדרה. כל תת סדרה מסתיימת כאשר אחר האיבר האחרון מופיע איבר שווה או קטן לו.
isTripleList (Node<int> L)	רשימה L תיקרא משולשת אם היא מקיימת את שלושת התנאים הבאים: (א) הרשימה אינה ריקה; (ב) מספר האיברים בה מתחלק ב-3; (ג) האיברים בשליש הראשון של הרשימה מכילים את אותם ערכים שמכילים האיברים בשליש השני ובשליש השלישי ובאותו הסדר. כתוב פעולה המקבלת רשימה ומחזירה אמת אם היא משולשת אחרת שקר.
addSort(Node<int> L, int n)	הפעולה מקבלת רשימה ממוינת של int ומוסיפה איבר עם הערך החדש במקום המתאים. אם הערך קיים ניתן להוסיפו לפני או אחרי החוליה השווה לו.

הפעולה מקבלת רשימה של מספרים שלמים ומחזירה רשימה חדשה ממוינת (הרשימה המקורית לא משתנה). ניתן וכדאי לעשות שימוש בפעולות קודמות.	sortList(Node<int> L)
--	-----------------------

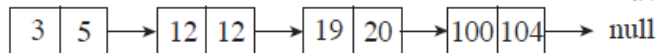
שאלה מבגרות 2010

2. L היא רשימה המכילה מספרים שלמים שונים זה מזה וממוינים בסדר עולה. **רשימת הטווחים** של L היא רשימה חדשה שנבנית באופן הזה: בעבור כל רצף של מספרים עוקבים ב-L יהיה ברשימת הטווחים איבר אחד שמכיל שני מספרים. מספר אחד הוא המספר הקטן ביותר ברצף, והמספר השני הוא המספר הגדול ביותר ברצף. רצף יכול להיות באורך 1 או יותר. אם הרצף הוא באורך 1, הוא מיוצג ברשימת הטווחים על ידי איבר ששני המספרים בו שווים.

לדוגמה, בעבור הרשימה L שלפניך:



רשימת הטווחים של L תהיה:



לפניך תיאור חלקי של המחלקה **RangeNode**, המייצגת איבר ברשימת הטווחים.

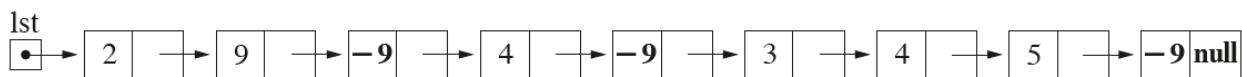
RangeNode	
private int from;	// המספר הקטן ביותר ברצף
private int to;	// המספר הגדול ביותר ברצף
public RangeNode(int from, int to)	

ממש ב-Java או ב-C# פעולה חיצונית שתקבל רשימה לא ריקה, המכילה מספרים שלמים שונים זה מזה וממוינים בסדר עולה, ותחזיר את רשימת הטווחים שלה.

```
public static Node<RangeNode> CreateRangeList(Node<int> sourceList)
```

שאלה מבגרות 2020

4. • "שרשרת מספרים שלמים חיוביים" היא שרשרת חוליות שכל חוליה בה מכילה מספר שלם הגדול מ-0.
- "שרשרת ספרות" היא שרשרת חוליות שכל חוליה בה מכילה ספרה בין 0 ל-9 (כולל) או את המספר (-9).
- כל רצף ספרות בשרשרת מייצג מספר: הספרה הראשונה מייצגת את האחדות, הספרה השנייה את העשרות וכן הלאה. לאחר כל רצף של ספרות מופיע המספר (-9), והוא מסמן סוף של מספר בשרשרת.
- לפניך דוגמה ל"שרשרת ספרות" המייצגת את המספרים: 92, 4, 543.



כתוב פעולה חיצונית בשפת Java בשם buildDigit או בשפת C# בשם BuildDigit, המקבלת הפניה lst שאינה null ל"שרשרת מספרים שלמים חיוביים". הפעולה תחזיר "שרשרת ספרות" המייצגת את המספרים שב"שרשרת מספרים שלמים חיוביים" לפי הסדר.

שאלה ברמת בגרות

ממשו את הפעולה הבאה ונתחו את יעילותה:

```
public static int maxSortedSubList(List<Integer> lst)
```

הפעולה מקבלת רשימה של מספרים שלמים ומחזירה את אורך התת-רשימה הרציפה הארוכה ביותר המסודרת בסדר עולה ממש.

דוגמאות:

- עבור הרשימה: $lst = 3, 1, 2, 3, 2, -3, -1, 2, 4, 7$, הזימון `maxSortedSubList(lst)` יחזיר את הערך 5. התת-רשימה העולה הארוכה ביותר, היא התת-רשימה המתחילה ב: -3 ומסתיימת ב: 7, ויש בה חמישה איברים.
- עבור הרשימה: $lst = 10, 7, 5, 3, -2, -30$, הזימון `maxSortedSubList(lst)` יחזיר את הערך 1, מכיוון שלא קיימת תת-רשימה עולה שאורכה 2 לפחות.
- עבור הרשימה: $lst = -4, 0, 1, 6, 12, 23, 90$, הזימון `maxSortedSubList(lst)` יחזיר את הערך 7. התת-רשימה העולה הארוכה ביותר היא כל הרשימה, ויש בה 7 איברים.

תור Queue

```

public class Queue <T>
{
    private Node<T> first;
    private Node<T> last;
    0 references
    public Queue()...
    0 references
    public bool isEmpty ()...
    0 references
    public T Head()...
    0 references
    public void Insert (T value) // מוסיף איבר לסוף התור...
    0 references
    public T Remove () // מוציא ומחזיר את האיבר הראשון בתור...
    0 references
    public override string ToString()...
}

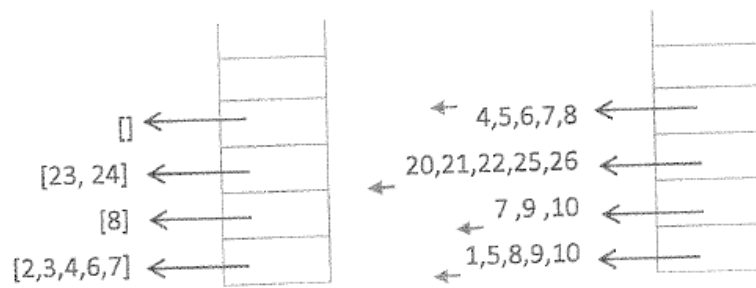
```

מהות	הפעולה
פעולה גנרית חיצונית המקבלת מערך של $\langle T \rangle$ ויוצרת ממנו תור.	CreateQueue $\langle T \rangle$
שירשור – פעולה גנרית המקבלת שני תורים q_1, q_2 ומשרשרת אותם לתור אחד q_1 (מחברת את q_1 עם q_2).	Concatenate $\langle T \rangle$
פעולה גנרית המקבלת תור וערך ובודקת אם הערך קיים בתור. התור אינו משתנה. שימוש בתור עזר	isExist1 $\langle T \rangle$
פעולה גנרית המקבלת תור של איברים שונים זה מזה וערך, ובודקת אם הערך קיים בתור. התור אינו משתנה ואין להשתמש במבנה נתונים נוסף.	isExists2 $\langle T \rangle$
פעולה המקבלת תור של מספרים חיוביים וערך, ובודקת אם הערך קיים בתור. התור אינו משתנה ואין להשתמש במבנה נתונים נוסף.	isExist3
פעולה גנרית חיצונית המקבלת תור ומחזירה העתק של תור. המחסנית המקורית אינה משתנה.	Copy
פעולה גנרית המקבלת תור והופכת את סדר אבריו.	Reverse $\langle T \rangle$
פעולה גנרית המקבלת תור של מספרים שלמים ומוציאה ומחזירה את המספר המינימלי. התור לא ישתנה למעט הוצאת המספר המינימלי. אם יש מספר מספרים מינימליים זהים הפעולה תוציא רק אחד מהם.	PopMin
כתבו פעולה המקבלת תור int ומחזירה תור חדש הממוין מהמספר הקטן לגדול (המספר הקטן בראש המחסנית). בסיום הפעולה התור המקורי לא משתנה.	Sort
כיתבו פעולה המקבלת תור של מספרים שלמים ומספר שלם n . הפעולה תחזיר $true$ אם מספר החזרות של כל אחד מהמספרים בטווח $1-n$ הוא כערך המספר עצמו : המספר 1 מופיע פעם אחת, המספר 2 מופיע פעמיים וכך הלאה. אין חשיבות לסדר המספרים. אין להשתמש באלגוריתם למיון. התור לא ישתנה. (שאלה 5 בספר).	Sequence
כיתבו פעולה חיצונית המקבלת תור המכיל מספרים טבעיים שונים זה מזה. על הפעולה להחזיר מחסנית ממוינת בסדר עולה (האיבר הגדול ביותר בראש המחסנית) הבנויה מכל אברי התור, תוך התחשבות במגבלות הבאות : (1) בשום שלב אין לשלוף איברים מהמחסנית ; (2) אין להשתמש במבני נתונים נוספים למעט התור שהתקבל.	SortedStack
כתבו פעולה חיצונית המקבלת שני תורים q_1, q_2 של מספרים ממשיים $double$ וממיינים מהקטן לגדול. הפעולה תמזג אותם לתוך q_1 כתור יחיד ממויין.	Merge

bestStudents	<p>כתבו פעולה המקבלת תור של תלמידים, ומחזירה מחסנית של התלמידים עם הציון הכי גבוה (יכולים להיות מספר תלמידים שקיבלו את הציון הכי גבוה). הערך הוא אובייקט Student הכולל string name ו-int grade. הניחו שיש פעולות get,set במחלקה ובנאים כפי שתבחרו. סיבוכיות הזמן חייבת להיות $O(n)$.</p>
--------------	---

תור של תורים ותור של שרשרת חוליות

נתון תור QQ שבו כל איבר הוא תור של מספרים שלמים. כל תור ממויין בסדר עולה כך שהאיבר בראש התור הוא הקטן ביותר.
לדוגמה:



- א. כיתבו פעולה חיצונית המקבלת תור של תורים QQ ומחזירה תור שכל איבר בו הוא שרשרת חוליות. QL הוא תור משלים לתור QQ כך שבכל שרשרת הממוקמת בתור QL נמצאים כל המספרים החסרים בתור הממוקם באותו מקום בתור QQ, כדי שיהיה רצף של מספרים עוקבים מהקטן ועד הגדול. על המספרים בשרשרות להיות ממוינים מהקטן לגדול כך שהמספר בתחילת השרשרת הוא הקטן ביותר.
- ב. מה סיבוכיות זמן ריצה של הפעולה שכתבתם? נמקו

שאלה מבגרות 2021

א. ממשו את הפעולה Size המתוארת בטבלה הבאה.

תיאור הפעולה	כותרת הפעולה
הפעולה מחזירה את מספר האיברים בתור q	בשפת Java — <code>public static int size (Queue<Integer> q)</code>
בלי לשנות את התור.	בשפת C# — <code>public static int Size (Queue<int> q)</code>

ב. שני תורים, q_1 ו- q_2 , יהיו "תורים זהים" אם מספר האיברים בשני התורים זהה, ובשני התורים מופיעים בדיוק אותם ערכים ובאותו הסדר.

דוגמה לשני תורים זהים:

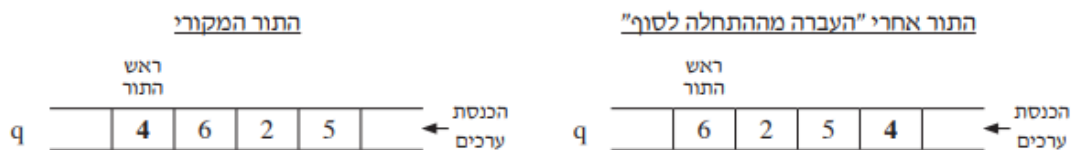


כתוב פעולה חיצונית ששמה `isIdentical` בשפת Java או `IsIdentical` בשפת C# המקבלת שני תורים מטיפוס שלם q_1 ו- q_2 , ומחזירה `true` אם התורים זהים, אחרת היא מחזירה `false`.

הערה: עם סיום הפעולה, חובה לשמור על מבנה התורים המקורי שהתקבל.

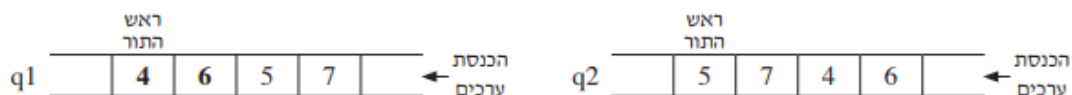
ג. "העברה מההתחלה לסוף" היא העברת מספר מראש התור לסופו.

דוגמה:



כתוב פעולה חיצונית ששמה `isSimilar` בשפת Java או `IsSimilar` בשפת C# המקבלת שני תורים מטיפוס שלם, q_1 ו- q_2 . הפעולה מחזירה `true` אם התורים q_1 ו- q_2 זהים — בין שהם זהים כמו שהם ובין שהם יהיו זהים לאחר שנבצע ב- q_1 "העברה מההתחלה לסוף", פעם אחת או יותר. אחרת הפעולה מחזירה `false`.

דוגמה: הפעולה תחזיר `true` בעבור שני התורים q_1 ו- q_2 שלפניך:



זאת מכיוון שלאחר שבתור q_1 נבצע פעמיים "העברה מההתחלה לסוף", הוא יהיה זהה לתור q_2 , וייראה כך:



הערות: — חובה להשתמש בפעולה שכתבת בסעיף א.

— אין צורך לשמור על מבנה התורים המקורי שהתקבל.

/המשך בעמוד 18/

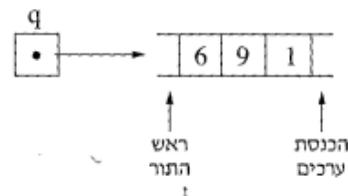
שאלה מבגרות 2019

5. א. "תור מספר" הוא תור של ספרות בין 1 ל-9 (כולל), המייצג מספר שלם – האיבר הראשון (ראש התור) הוא

ספרת האחדות, האיבר השני הוא ספרת העשרות וכן הלאה.

הנח שמספר הספרות האפשרי בתור לא גדול ממספר הספרות שיכול להכיל משתנה מטיפוס `int`.

לדוגמה: התור שלפניך מייצג את המספר 196.



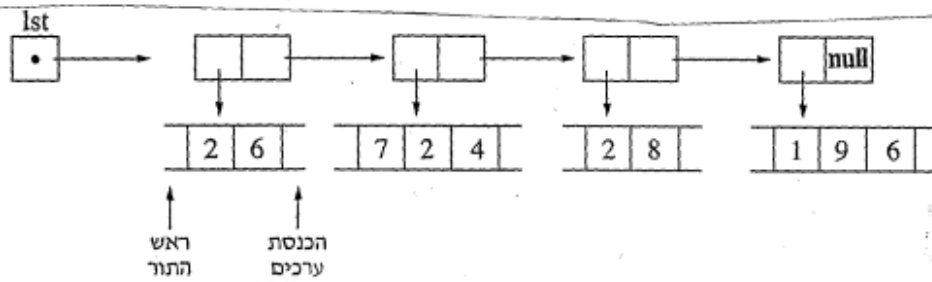
כתוב פעולה בשם `toNumber` ב-Java או `ToNumber` ב-C#, המקבלת "תור מספר" – `q`, ומחזירה

את המספר המיוצג בתור.

הערה: אין חובה לשמור על מבנה התור.

ב. נתונה שרשרת חוליות ובה כל חוליה מכילה "תור מספר".

לדוגמה: שרשרת החוליות שלפניך מייצגת את המספרים 62, 427, 82, 691.



כתוב פעולה בשם `bigNumber` ב-Java או `BigNumber` ב-C#, המקבלת הפניה `lst` לשרשרת החוליות,

ומחזירה את המספר הגדול ביותר המיוצג בשרשרת החוליות.

עבור שרשרת החוליות שתוארה בדוגמה הפעולה מחזירה את המספר 691.

חובה להשתמש בפעולה שהוגדרה בסעיף א.

שאלה מבגרות 2006

1. לפניך פעולה:

<p>תור_לפי_שכיחות (Q)</p> <p>הפעולה מקבלת תור Q המכיל מספרים שלמים, ומחזירה תור חדש. בעבור כל מספר בתור Q, יהיו בתור החדש שני איברים: האיבר הראשון מכיל את המספר מהתור Q, והאיבר השני מכיל את מספר הפעמים שהוא מופיע בתור Q. בעבור מספר המופיע יותר מפעם אחת בתור Q, יהיה זוג אחד בלבד בתור החדש.</p> <p>הנחה: התור Q מאותחל.</p> <p>הערה: אין לשנות את התור Q.</p>	
--	--

לדוגמה:

נתון התור Q (משמאל לימין):

1	4	4	1	5	-9	1	-9	-9
---	---	---	---	---	----	---	----	----

התור שיוחזר לאחר זימון הפעולה **תור_לפי_שכיחות (Q)** יהיה (משמאל לימין):

1	3	4	2	5	1	-9	3
---	---	---	---	---	---	----	---

א. כתוב אלגוריתם, שיממש את הפעולה **תור_לפי_שכיחות (Q)**.

אפשר להשתמש בפעולות הממשק תור ובפעולה **העתק_תור (Q)** שלפניך,

בלי לממש אותן.

<p>העתק_תור (Q)</p> <p>הפעולה מקבלת תור Q ומחזירה תור חדש זהה לו.</p> <p>הנחה: Q מאותחל.</p> <p>סיבוכיות זמן הריצה: $O(n)$, n הוא מספר האיברים בתור Q.</p>	
--	--

ב. מהי סיבוכיות זמן הריצה של האלגוריתם שכתבת בסעיף א? נמק את תשובתך.

הנח שסיבוכיות זמן הריצה של כל אחת מפעולות הממשק תור היא $O(1)$.

/המשך בעמוד 3/

+

שאלה מבגרות 2009 – רקורסיה

לפניך 2 פעולות חיצוניות הכתובות ב- C#.

הפעולה מקבלת תור לא ריק, המכיל מספרים שלמים.
 /** הפעולה מחזירה... */

```
public static int Sod1 (Queue<int> q)
{
    int i = q.Remove();
    int result = i;

    if (!q.IsEmpty())
    {
        int j = Sod1 (q);
        if (result > j)
            result = j;
    }
    q.Insert(i);
    return result;
}
```

הפעולה מקבלת מספר שלם גדול מ- 0 או שווה לו
 /** הפעולה מחזירה... */

```
public static int Sod2 (int i)
{
    if (i == 0)
        return 0;
    int a = i % 10;
    int b = Sod2(i / 10);
    if (a > b)
        return a;
    return b;
}
```

עצים בינאריים

```

public class BinNode <T>
{
    private BinNode<T> left;
    private T value;
    private BinNode<T> right;

    0 references
    public BinNode (T value) [...]
    0 references
    public BinNode (BinNode<T> left, T value, BinNode<T> right) [...]

    0 references
    public T GetValue () { return this.value; }
    0 references
    public BinNode<T> GetLeft () { return this.left; }
    0 references
    public BinNode<T> GetRight () { return this.right; }
    0 references
    public bool HasLeft () { return this.left != null; }
    0 references
    public bool HasRight () { return this.right != null; }
    0 references
    public void SetValue (T value) { this.value = value; }
    0 references
    public void SetLeft (BinNode<T> left) { this.left = left; }
    0 references
    public void SetRight (BinNode<T> right) { this.right = right; }
    0 references
    public override string ToString() {return this.value.ToString();}
}

public static void graphicTree ()
{
    BinNode<int> t = BinTreeUtils.BuildRandomTree(50, 1, 3);
    TreeCanvas.AddTree(t);
    TreeCanvas.TreeDrawPostOrder();
}

```

מחלקה גרפית לייצוג עץ בינרי

הקובץ *Unit.dll* מכיל את מחלקת השירות *TreeCanvas* במרחב השמות *Unit4.CanvasLib*.

מחלקת השירות מאפשרת הצגה גרפית של עצים בינריים המבוססים על המחלק הגרפית *BinNode<T>*, וכן מוצגות הסריקות הבאות לפי העץ שהתקבל: סריקה תחילית, סריקה תוכית, סריקה סופית וסריקה לפי רמות.

<code>static void AddTree<T>(BinNode<T> tree)</code>	הפעולה מעבירה את העץ הבינרי המתקבל כפרמטר לחלון גרפי וגם מציגה אותו.
<code>static void SetAnimationSpeed(AnimationSpeed speed)</code>	הפעולה קובעת מהירות של אנימציה בסריקה של העץ הבינרי שהועבר לחלון גרפי על-ידי הפעולה <i>AddTree</i> .
<code>static void PrintOutAllElements(bool printOut)</code>	הפעולה קובעת האם להציג תוצאות הסריקה בחלון <i>Console</i> , של העץ הבינרי שהועבר לחלון גרפי על-ידי הפעולה <i>AddTree</i> .
<code>static void TreeDrawInOrder()</code>	פעולה מתארת את סריקת העץ הבינרי שהועבר לחלון גרפי בסדר תוכי.
<code>static void TreeDrawPostOrder()</code>	פעולה מתארת את סריקת העץ הבינרי שהועבר לחלון גרפי בסדר סופי.
<code>static void TreeDrawPostOrder()</code>	פעולה מתארת את סריקת העץ הבינרי שהועבר לחלון גרפי בסדר תחילי.
<code>static void TreeDrawLevelOrder()</code>	פעולה מתארת את סריקת העץ הבינרי שהועבר לחלון גרפי לפי רמות.

```

/***** Tree Print*****/
public static void BinTreeMethods()
{
    string path = GetTreeFilePath("tree_new.txt");
    BinNode<int> tree = BuildBinaryTree<int>(path);
    PrintBinaryTree(tree);
    PrintBinaryTreeColored(tree);
}

```

```

tree_new.txt
1      10
2      Left:3
3      Left:4
4      Left: 0
5      Right: 2
6      Right:6
7      Left:1
8      Right:27
9      Right: 4
10     Left:2
11     Right:3
12

```

סדר הסריקה לעומק DFS

```
public static void DFS <T> (BinNode<T> node)
{
    if (node == null)
        return;
    Console.Write(node + ", ");
    DFS(node.GetLeft());
    DFS(node.GetRight());
}
```

```
public static void DFS <T> (BinNode<T> node)
{
    if (node == null)
        return;
    // preOrder - תחילית
    DFS(node.GetLeft());
    // inOrder - תוכנית
    DFS(node.GetRight());
    // postOrder - סופית
}
```

תיאור	הפעולה
התקן את הממשקים הנדרשים. צור קובץ עם עץ <code><int></code> בעל שלוש רמות לבחירתך. הדפס את העץ באמצעות הספריה הגרפית.	
פעולה המקבלת עץ (שורש - חוליה בינרית) ומחזירה תור עם סדר הערכים בסריקה תחילית	DFS
הפעולה סופרת את מספר הצמתים בעץ.	DFS_count
הפעולה מחזירה את ערך הצומת הגבוה ביותר בעץ.	DFS_FindMax
הפעולה מחזירה את ערך העלה הגבוה ביותר בעץ.	DFS_FindMaxLeaf
הפעולה מחזירה את עומק העץ	DFS_depth
הפעולה סוכמת את כל הערכים של צמתי העץ.	DFS_sum_of_nodes
הפעולה סוכמת את כל הערכים של עלי העץ.	DFS_sum_of_leaves
הפעולה מחזירה true אם כל העלים זוגיים	DFS_IsAllEven
הפעולה מקבלת עץ ומסדרת את כל העלים כך שהעלה הקטן יהיה מצד ימין.	DFS_SortSons

בגרות 2022

7. שאלה בנושא עץ בינארי

בשאלה זו אפשר להשתמש בפעולה החיצונית `eraseFirst / EraseFirst` שלהלן בלי לממש אותה.

דוגמאות	תיאור הפעולה	כותרת הפעולה
<ul style="list-style-type: none"> – בעבור המחרוזת <code>str = "hello"</code>, הפעולה תחזיר את המחרוזת <code>"ello"</code>. – בעבור המחרוזת <code>str = "temp"</code>, הפעולה תחזיר את המחרוזת <code>"emp"</code>. – בעבור המחרוזת <code>str = "m"</code>, הפעולה תחזיר מחרוזת ריקה <code>""</code>. – בעבור המחרוזת הריקה <code>str = ""</code>, תהיה שגיאה. 	<p>הפעולה מחזירה תת־מחרוזת של <code>str</code>, ללא התו הראשון. אם המחרוזת <code>str</code> ריקה לפני זימון הפעולה, תהיה שגיאה.</p>	<p>בשפת Java: <code>public static String eraseFirst (String str)</code></p> <p>בשפת C#: <code>public static string EraseFirst (string str)</code></p>

ממשו את הפעולה החיצונית שלהלן:

Java – `public static boolean wordFromRoot (BinNode<Character> tree, String str)`

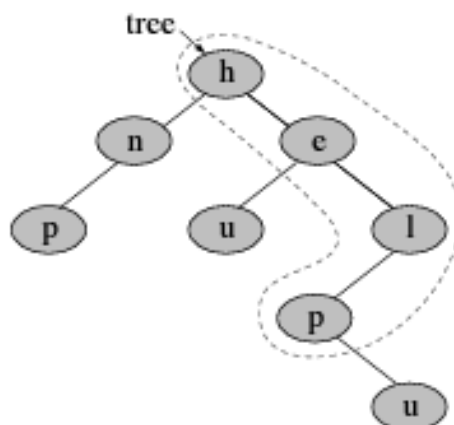
C# – `public static bool WordFromRoot (BinNode<char> tree, string str)`

הפעולה מקבלת מחרוזת `str` – המכילה לפחות תו אחד, והפניה לעץ בינארי של תווים `tree` שאינו `null`. הפעולה תחזיר `true` אם קיים מסלול המתחיל בשורש העץ שבו בצף התווים זהה למחרוזת `str`. אחרת הפעולה תחזיר `false`.

הערה: אות קטנה ואות גדולה אינן זהות זו לזו.

דוגמה:

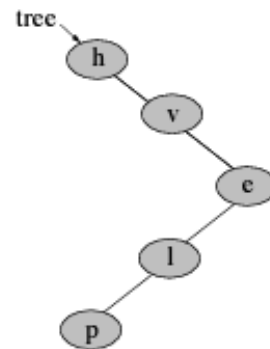
בעבור העץ הנתון והמחרוזת `"help"` הפעולה תחזיר `true`.



(שימו לב: המשך השאלה בעמוד הבא.)

דוגמה:

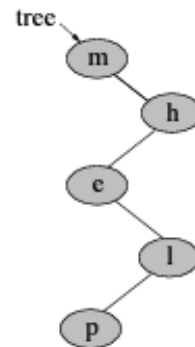
בעבור העץ הנתון והמחרוזת "help" הפעולה תחזיר false.



הסבר: לא קיים בעץ רצף תווים הזהה למחרוזת "help".

דוגמה:

בעבור העץ הנתון והמחרוזת "help" הפעולה תחזיר false.

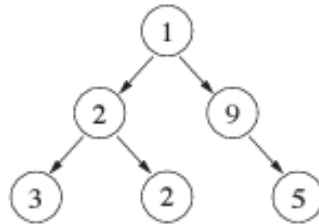


הסבר: אף על פי שקיים בעץ רצף תווים הזהה למחרוזת "help", הפעולה תחזיר false, כי הרצף אינו מתחיל בשורש העץ.

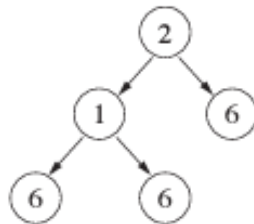
בגרות 2020

6. נגדיר: "עץ מספרים" הוא עץ בינארי מטיפוס שלם, שכל צומת בו מכיל ספרה בין 1 ל-9 (כולל), וכל מסלול בעץ מן השורש לעלה מייצג מספר: העלה מייצג את ספרת האחדות, הרמה שמעליו את ספרת העשרות וכן הלאה עד השורש של העץ.

דוגמה: בעץ המספרים שלפניך מיוצגים המספרים: 123, 122, 195 (במסלולים בעץ משמאל לימין).



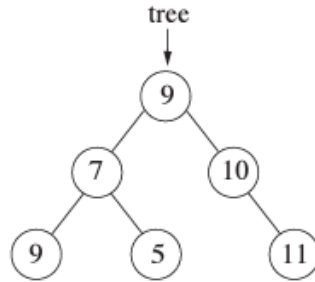
דוגמה נוספת: בעץ המספרים שלפניך מיוצגים המספרים: 216, 216, 26 (במסלולים בעץ משמאל לימין).



כתוב פעולה חיצונית `printAll` בשפת Java או `PrintAll` בשפת C#. הפעולה תקבל עץ מספרים `tree` מטיפוס שלם ותדפיס את כל המספרים שהמסלולים בעץ מייצגים.
 אם `tree` הוא `null` הפעולה לא תדפיס דבר.
הערה: אין חשיבות לסדר שבו המספרים מודפסים.

בגרות 2020 – מועד מיוחד

6. עץ בינרי מטיפוס שלם של מספרים שאינם שליליים הוא "עץ שאריות שוויוני" במקרה הזה:
 כמות האיברים שמספריהם מתחלקים ב-3 עם שארית 1 שווה לכמות האיברים שמספריהם מתחלקים ב-3 עם שארית 2, ושווה לכמות האיברים שמספריהם מתחלקים ב-3 ללא שארית.
 דוגמה של "עץ שאריות שוויוני":

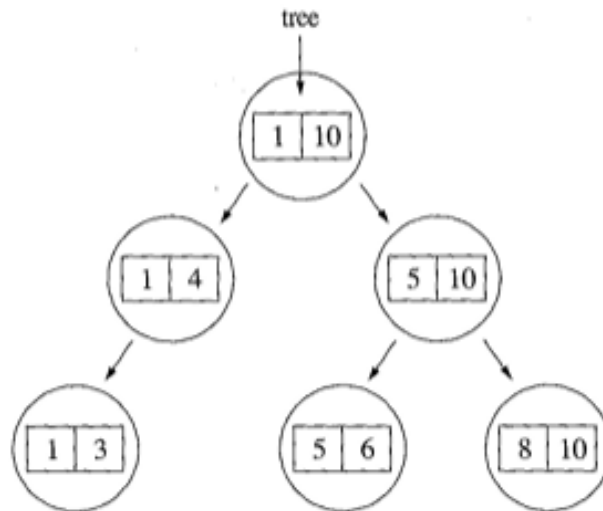


עץ בינרי זה הוא "עץ שאריות שוויוני" משום שיש בו שני מספרים שמתחלקים ב-3 ללא שארית (9,9), שני מספרים שמתחלקים ב-3 עם שארית 1 (10,7) ושני מספרים שמתחלקים ב-3 עם שארית 2 (11,5).

כתוב פעולה חיצונית בוליאנית בשפת Java בשם `treeEqual` או בשפת C# בשם `TreeEqual` המקבלת עץ בינרי מטיפוס שלם, לא ריק, של מספרים שאינם שליליים ובודקת אם הוא "עץ שאריות שוויוני".
 אם כן – תחזיר הפעולה `true`, אחרת היא תחזיר `false`.

בגרות 2019

דוגמה לעץ טווחים מסודר:



כתוב פעולה חיצונית בוליאנית בשם `order` ב-Java או `Order` ב-C#, המקבלת עץ טווחים או עץ ריק ומחזירה `true` אם העץ הוא עץ טווחים מסודר, אחרת – הפעולה מחזירה `false`.

6. נתונה המחלקה **Range** שיש לה שתי תכונות:

`low` – מספר מטיפוס שלם.

`high` – מספר מטיפוס שלם.

`high` גדול מ-`low`.

הנח שלכל תכונה הוגדרו ב-Java הפעולות `get` ו-`set` וב-C# הפעולות `Get` ו-`Set`.

עץ טווחים הוא עץ שאיבריו הם מטיפוס **Range**.

עץ טווחים מסודר הוא עץ ריק או עץ טווחים שבו עבור כל צומת מתקיימים התנאים האלה:

- אם יש בן שמאלי, אז ה-`low` של הצומת שווה ל-`low` של הבן השמאלי, וה-`high` של הצומת גדול או שווה ל-`high` של הבן השמאלי.
- אם יש בן ימני, אז ה-`high` של הצומת שווה ל-`high` של הבן הימני, וה-`low` של הצומת קטן או שווה ל-`low` של הבן הימני.
- אם יש שני בנים, אז ה-`high` של הבן השמאלי קטן מה-`low` של הבן הימני.

סריקה לעומק ללא רקורסיה

```
public static void DFS_Stack <T> (BinNode<T> node)
{
    Stack<BinNode<T>> stack = new Stack<BinNode<T>>();
    stack.Push(node);
    while (!stack.IsEmpty())
    {
        node = stack.Pop();
        Console.Write(node + ", ");
        if (node.HasRight()) { stack.Push(node.GetRight()); }
        if (node.HasLeft()) { stack.Push(node.GetLeft()); }
    }
    Console.WriteLine();
}
```

BFS - סריקה לרוחב ללא רקורסיה

```
public static void BFS<T>(BinNode<T> node)
{
    Queue<BinNode<T>> queue = new Queue<BinNode<T>>();
    queue.Insert(node);
    while (!queue.IsEmpty())
    {
        node = queue.Remove();
        Console.Write(node + ", ");
        if (node.HasLeft()) { queue.Insert(node.GetLeft()); }
        if (node.HasRight()) { queue.Insert(node.GetRight()); }
    }
    Console.WriteLine();
}
```