

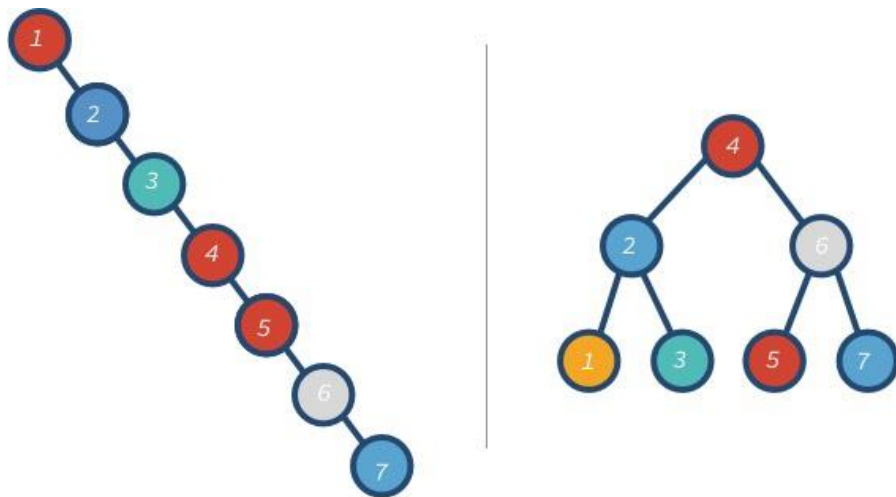
בונוס - תרגול 9 עץ AVL מטומפלט

רקע

הבונוס הבא הוא לפחות באורך של תרגיל רגיל.
מומלץ לעשות אותו לאורך כמה שבועות, במקביל לתרגילי הכיתה.

מטרה

בתרגיל הבונוס נממש עץ AVL גנרי.
עץ AVL הוא עץ שמאזן את עצמו ככה שחיפוש אלמנט בעץ ידרוש לכל היותר $\log(n)$ פעולות.



Non-Balanced Tree

vs

Balanced Tree

אלה השלבים שנעבור:

למידה עצמית	יצירת מחלקת AVLNode	מימוש מתודת insert	בנוס – מימוש מתודת remove
<ul style="list-style-type: none"> למידה על עץ AVL למידה על פעולות איזון של העץ (סיבובים) 	<ul style="list-style-type: none"> יצירת מחלקת AVLNode התאמת הקוד של BNode 	<ul style="list-style-type: none"> מימוש מתודת הכנסת אלמנט לעץ AVL מימוש rotations (RR, RL, LR, LL) 	<ul style="list-style-type: none"> מימוש מתודת הסרת אלמנט מעץ AVL

נתרגל מיומנויות חשובות:

- שימוש רקורסיה
- מימוש מבני נתונים מתקדמים
- יצירת מחלקות templates
- למידה עצמית!
- את התרגיל צריך להגיש ב-GIT: [לינק להוראות שימוש ב-GIT](#). כדאי לקרוא גם [דגשים לתכנות נכון](#).



"PRACTICE MAKES PERFECT"

בהצלחה יא אלופות ואלופים!

שלב 1: למידה עצמית - AVL

שלב בונס מימוש מתודת remove	מימוש מתודת insert	יצירת מחלקת AVLNode	למידה עצמית
-----------------------------------	-----------------------	------------------------	-------------

עץ AVL (Adelson Velskii Landis)

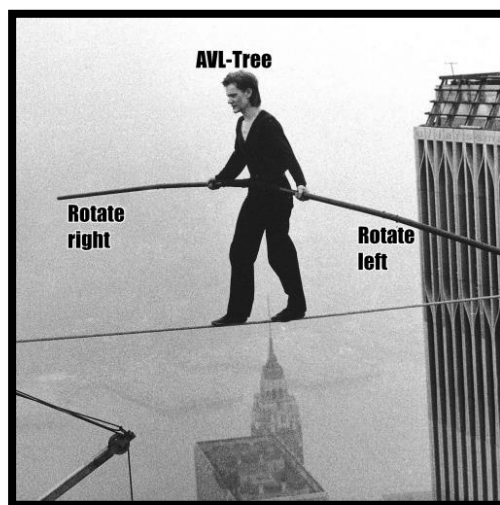
היזכרו במה שלמדנו בסוף שיעור 9 על **עצי AVL**, תוכלו לרענן את הידע שלכם/ן ע"י צפייה בסרטון הבא:
https://www.youtube.com/watch?v=vRwi_UcZGjU

כדי להתנסות באופן יותר ויזואלי תוכלו גם להשתמש באתר הבא:
<https://visualgo.net/en/bst?mode=AVL>

נסו להוסיף ולמחוק אלמנטים מהעץ ותראו איך הוא מגיב בהתאם.

בנוסף, אם הרגשתם/ן שהסרטון והחומר של השיעור לא מספיק ברורים, תוכלו להיעזר בסרטונים הבאים שמסבירים על עצי AVL בצורה **מאוד מקיפה ויסודית**.

- https://www.youtube.com/watch?v=-9sHvAnLN_w
- https://www.youtube.com/watch?v=rwzuze_tTwQ
- <https://www.youtube.com/watch?v=ONsmm7pTf2M>
- <https://www.youtube.com/watch?v=z-glRPMefxU>



שלב 2: יצירת מחלקת AVLNode

שלב בונס מימוש מתודת remove	מימוש מתודת insert	יצירת מחלקת AVLNode	למידה עצמית
-----------------------------------	-----------------------	------------------------	-------------

סעיף א' – עדכון המחלקה BSNode

לפני שנוכל באמת להתחיל, צריך לעדכן כמה מתודות של המחלקה BSNode

- עדכנו את המחלקה כך שהמתודה insert תחזיר מצביע ל-root של העץ (או תת העץ שעליו פועלת הפעולה).
- עדכנו את הפונקציה insert כך שהיא ניתנת לדריסה.
- עדכנו את ה-DTOR כך שיתמוך בהורשה.
- שנו את הגדרת השדות כך שיהיו נגישים למחלקות יורשות.

שימו 

על מנת שתוכלו להשתמש בגרפיקה של העץ מהסעיף הקודם, יש לבצע שינוי קל.

יש לכתוב את הקובץ printTreeToFile (הצהרה) כך שיתמוך בהוספת הטמפלייטים ובשינוי שעשינו בסעיף א'. ממשו את הפונקציות בקובץ ההצהרה.
לשם כך עליכם ללמוד איך לכתוב בצורה בסיסית לקבצים ב-C++.
בדקו שאכן הכל תקין.

כדי לממש את $< \text{operator} =$ אפשר לנסות לבד, ואם נתקעים אפשר למשל לחפש בגוגל `implement < operator`, להיכנס לקישור השני ולגלול לאזור המתאים.

כדי לממש את $<<$ אפשר להיעזר בתרגילים 3 (הבונס), 4 או לחפש בגוגל `implement operator <<`

סעיף ב' – מחלקת AVLTree

ממשו את הפונקציות הפרטיות הבאות במחלקה AVLTree, אשר יורשת מ-BSNode

- getBalanceFactor - מחשבת ומחזירה את "גורם האיזון" של הצומת. מחזירה מספר שלם.
(balanceFactor = leftChildHeight - rightChildHeight)
 - rotateRight - מבצעת רוטציה * ימנית עבור אותו עץ (או תת-עץ) שהשורש שלו הוא הצומת עליו הופעלה הפעולה.
הפונקציה תחזיר את הצומת שכרגע מייצגת את השורש של אותו תת עץ, בעקבות השינוי(הרוטציה).
 - rotateLeft - מבצעת רוטציה * שמאלית עבור הצומת שהשורש שלו הוא הצומת עליו הופעלה הפעולה.
הפונקציה תחזיר את הצומת שכרגע מייצגת את השורש של אותו תת עץ, בעקבות השינוי(הרוטציה).
 - * אין סיבה לבדוק האם יש צורך ברוטציה או לא. הפונקציה הזאת פשוט מבצעת רוטציה. (מי שיקרא לה ידאג לוודא שבאמת צריך לבצע רוטציה).
- בידקו שהפונקציות עובדות כראוי. ניתן להיעזר בגרפיקה. + נתון לכם קובץ בדיקה. היעזרו בו.



שלב 3: מימוש מתודת insert

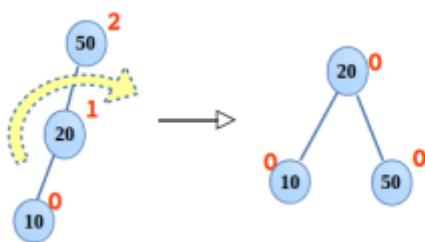
למידה עצמית	יצירת מחלקת AVLNode	מימוש מתודת insert	שלב בונס מימוש מתודת remove
-------------	---------------------	--------------------	-----------------------------

כעת נדרוס את הפונקציה insert.

1. **ערכים כפולים** - במידה והערך שמנסים להכניס כבר קיים בעץ נזרק `exception` מסוג `invalid_argument`.
2. **הפרות איזון** – במידה וזוהתה הפרה באיזון העץ נטפל בה באמצעות סיבוב העץ (rotation), נפריד למקרים LL, LR, RL, RR, ובצע סיבובים לפי הצורך.
כל הפונקציות המטפלות בהפרות איזון צריכות להחזיר מצביע לשורש של העץ (תת-עץ) שהשורש שלה הוא הצומת שעליו פעלה הפעולה.
למה? משום שהשורש יכול להשתנות במהלך הפונקציה בעקבות הרוטציה...
(עם זאת יכול להיות שהשורש המקורי נשאר השורש של העץ).

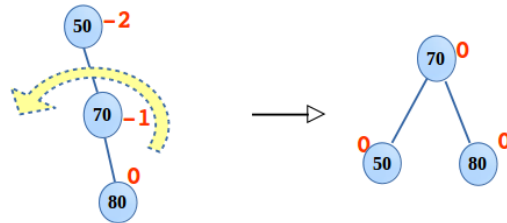
מקרה א' – הפרת RR

נטפל במקרה RR. זכרו שיתכן שהכנסה של צומת כלשהו לתוך תת-עץ יכולה ליצור הפרה בשורש של אותו תת-עץ.



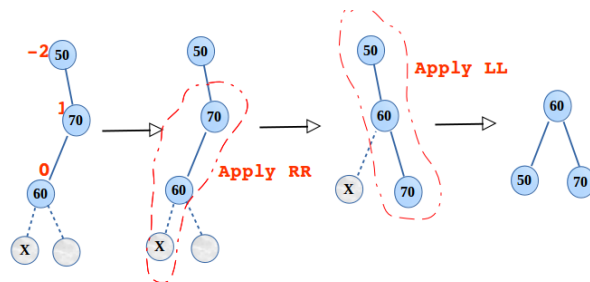
מקרה ב' – הפרת LL

נטפל במקרה LL. זכרו שיתכן שהכנסה של צומת כלשהו לתוך תת-עץ יכולה ליצור הפרה בשורש של אותו תת-עץ.



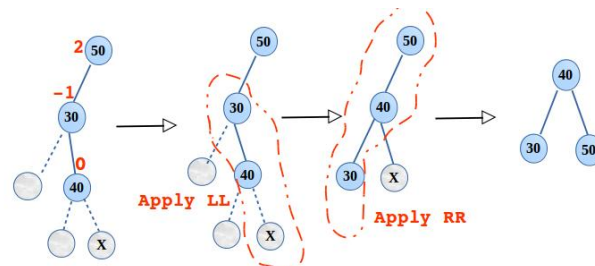
מקרה ג' – הפרת LR

נטפל במקרה LL. זכרו שיתכן שהכנסה של צומת כלשהו לתוך תת-עץ יכולה ליצור הפרה בשורש של אותו תת-עץ.




מקרה ד' – הפרת RL

נטפל במקרה LL. זכרו שיתכן שהכנסה של צומת כלשהו לתוך תת-עץ יכולה ליצור הפרה בשורש של אותו תת-עץ.



הנחיות והערות חשובות:

- חשוב שגם ההגדרות וגם המימוש יהיו באותו הקובץ (קובץ ה-`hpp` או ה-`h`), אחרת יהיו שגיאות לינקג'.
- ניתן להשתמש במילה השמורה `typename` או `class`, לנו זה לא משנה
- שימו , יתכן שהכנסה של צומת כלשהו לתוך תת-עץ יכולה ליצור הפרה בשורש של אותו תת-עץ. יתכן שתיקון מסוים ישפיע במעלה העץ (כלומר יצור הפרה באחד השורשים שמעליו)
- חישבו מתי מקרה כזה מתקיים.
- חישבו כמה רוטציות יש לעשות כדי לתקן את אותה צומת שגורם האיזון שלה הופר.
- חישבו אילו רוטציות יש לבצע.
- חישבו האם הבנים של אותה צומת משתנים או לא
- חישבו האם השורש של אותו תת-עץ השתנה או לא.
- בידקו היטב היטב!!!!

שלב בונוס: מימוש מתודת insert

שלב בונוס מימוש מתודת remove	מימוש מתודת insert	יצירת מחלקת AVLNode	למידה עצמית
------------------------------------	-----------------------	------------------------	-------------



בונוס לבונוס? יש דבר כזה? מסתבר שכן... 🐱

מחיקת אלמנט מהעץ

אם תרצו לאתגר את עצמכם/ן אפילו יותר,
טפלו במקרה של מחיקת ערך כלשהו מהעץ (remove).




בהצלחה!

נספחים

הגשה ב-GIT

- את הפרויקט יש לנהל ב-Git, לפתוח repository חדש בתוך קבוצת ה-gitlab שלנו ושל המדריך/ה, ולהגיש לינק לפרויקט ב-NEO (אפשר לעשות comment עם הלינק או להגיש מסמך txt עם הלינק בפנים).
- יש להעלות ל-repository את כל הקבצים הרלבנטיים לתרגיל (קבצי txt, מסמכים, ומשאבים אחרים שבהם השתמשנו).
- חשוב להעלות את פרויקט ה-Visual Studio השלם ולהתעלם מקבצים לא נחוצים ([הנחיות במסמך הבא](#)), במידה ולא הועלה הפרויקט השלם, אין להעלות את שאר הקבצים שיוצר Visual Studio – הם רבים מאוד, הם לא מכילים מידע נחוץ להרצת הפרויקט אצל המדריך, ורק יוצרים בלגן.
- הבחירה אילו קבצים להעלות ל-repository נעשית באמצעות הפקודות add ו-rm. אופציה נוספת (מומלצת) היא להוסיף קובץ gitignore. אשר יתעלם מהקבצים הלא נחוצים. במידה ותרצו תוכלו להיעזר ב[סרטוני עזר בנושא GIT](#).
- כסיימתם/ן, בדקו שניתן להריץ את הפרויקט בקלות – בצעו Clone אל תיקייה במחשב אשר שונה מזו שעבדתם/ן, ותראו שהפרויקט נפתח ע"י לחיצה על קובץ ה-sln ויכול לרוץ בלי בעיה

 Ex9Bonus.sln

דגשים:

- את הפרויקט יש לפתוח בקבוצת ה-gitlab שאליה משותף/ת המדריך/ה כ-Maintainer.
- יש לוודא שכל הקבצים הרלבנטיים נוספו ל-repository (באמצעות הפקודה add), במידת הצורך ניתן להוריד קבצים מיותרים (באמצעות הפקודה rm).
- יש לבצע commit עבור כל סעיף, ובנקודות שבהן הוספנו שינויים חשובים (לפי הדגשים שהועברו בכיתה).
- עבור כל commit, זכרו לכתוב הודעה קצרה ואינפורמטיבית, שאפשר יהיה להבין מה היה השינוי בקוד.
- יש לדחוף את הקוד (באמצעות הפקודה push) ל-repository בסיום העבודה שלנו, חשוב שבסיום העבודה שלנו, ובמידה ונפנה למדריך/ה, ב-repository יהיה הקוד המעודכן ביותר.
- במידה ושכחנו או שאנחנו לא בטוחים איך מעלים קובץ, או מתעלמים מקבצים, כדאי לצפות בסרטוני ההדרכה בנושאי GIT. ניתן לגשת לסרטונים בלשונית ה-resources שבכיתה ה-NEO.
- בסיום העבודה יש להגיש לכיתה ה-NEO קישור ל-repository.

כללי

1. יש לבדוק שכל המטלות מתקמפלות ורצות ב-VS2022. מטלה שלא תעבור קומפילציה אצל הבודק לא תיבדק **והניקוד שלה יהיה 0** 😞
2. יש לבדוק שהקוד שכתבתם עובד. יש להריץ בדיקות שלכם ולוודא שהקוד ברמה טובה.
3. כאשר אתם מתבקשים לממש פונקציה, ממשו בדיוק את הנדרש. אין להוסיף הדפסות וכדו'. אם הוספתם תוך כדי הבדיקות שלכם הדפסות, אנא דאגו להוריד אותם לפני ההגשה.
4. להזכירכם! העבודה היא עצמית, ואין לעשות אותה ביחד.
5. על כל שאלה או בעיה יש לפנות למדריך, לפחות 36 שעות לפני מועד ההגשה.

דגשים לתכנות נכון

- כדאי לקמפל כל מספר שורות קוד ולא לחכות לסוף! הרבה יותר קל לתקן כאשר אין הרבה שגיאות קומפילציה. בנוסף קל יותר להבין מאיפה השגיאות נובעות.
- כדאי לכתוב פונקציה ולבדוק אותה לפני שאתם ממשיכים לפונקציה הבאה. כלומר, כתבו תכנית ראשית שמשתמשת בפונקציה ובודקת האם היא עובדת כראוי. חישבו על מקרי קצה ונסו לראות מה קורה.
- בכל פעם שאתם מתקנים משהו, זכרו שיכול להיות שפגעתם במשהו אחר. לכן עליכם לבדוק שוב מהתחלה.
- חשפו החוצה רק את הממשק המינימלי הדרוש (minimal API), הגדירו את שדות המחלקה כפרטיים, וכמה שפחות מתודות כציבוריות.