

## מחלקת העל – WAVLTREE

שדות			
	מצביע לצומת וירטואלי, אקסטרני (לשימוש פנימי בלבד). מאותחל סופית. מחוסר הורה וילדים, ובעל דרגה 1.	WAVLNode EXT	private final
	מצביע לשורש העץ	WAVLNode root	Private
	מצביע לאיבר עם המפתח הקטן ביותר בעץ	WAVLNode max	Private
	מצביע לאיבר עם המפתח הגדול ביותר בעץ	WAVLNode min	Private
מתודות			
<b>בנאי ראשון</b>		<ul style="list-style-type: none"> <li>קלט: אין.</li> <li>יוצר עץ ריק חדש, כאשר לאחר האתחול השורש הוא צומת אקסטרני (לשימוש עתידי).</li> <li>סיבוכיות: <math>O(1)</math>.</li> </ul>	
<b>בנאי שני</b>		<ul style="list-style-type: none"> <li>קלט: מפתח מסוג int, מידע מסוג string.</li> <li>מאתחל את השורש על ידי יצירת עצם מסוג WAVLNode כאשר: <ul style="list-style-type: none"> <li>מפתח מתקבל מהקלט.</li> <li>מידע מתקבל מהקלט.</li> <li>הורה – EXT, הצומת האקסטרני (לשימוש עתידי).</li> </ul> </li> <li>משנה את המצביעים min, max לשורש (מהיותו היחיד בעץ).</li> <li>סיבוכיות: <math>O(1)</math>.</li> </ul>	
<b>Empty</b>		<ul style="list-style-type: none"> <li>קלט: אין.</li> <li>פלט: ערך בוליאני, האם העץ ריק או לא.</li> <li>שיטת פעולה: <ul style="list-style-type: none"> <li>במידה והשורש מצביע לצומת האקסטרני, EXT, אזי העץ ריק ומוחזר הערך true, על פי לוגיקת הבנאי הראשון.</li> <li>אחרת, מוחזר false.</li> </ul> </li> <li>סיבוכיות: <math>O(1)</math>.</li> </ul>	
<b>Search</b>		<ul style="list-style-type: none"> <li>קלט: מפתח מסוג int.</li> <li>פלט: מחרוזת המייצגת את המידע של הצומת עם המפתח k.</li> <li>שיטת פעולה: <ul style="list-style-type: none"> <li>במידה והעץ ריק, על ידי שימוש במתודה empty, מוחזר null.</li> <li>במידה ולא ריק, מבצעת חיפוש בינארי עד אשר מגיעה לצומת עם מפתח הקלט, במידה ומצא – מחזיר את המידע ממנו.</li> <li>אם הגענו עד לעלה חיצוני, EXT, סימן שהמפתח לא נמצא בעץ, ועל כן יוחזר null.</li> </ul> </li> <li>סיבוכיות: <math>O(\log n)</math>.</li> </ul>	
<b>Min</b>		<ul style="list-style-type: none"> <li>קלט: אין.</li> <li>פלט: מחרוזת המייצגת את המידע של המידע של הצומת עם המפתח המינימלי.</li> <li>שיטת פעולה: <ul style="list-style-type: none"> <li>במידה והעץ לא ריק, על ידי שימוש במתודה empty, פונה לצומת בעל המפתח המינימלי בעזרת השדה min, ומחזיר את המידע ממנו.</li> <li>אחרת, מחזיר null.</li> </ul> </li> <li>סיבוכיות: <math>O(1)</math>.</li> </ul>	
<b>Max</b>		<ul style="list-style-type: none"> <li>קלט: אין.</li> <li>פלט: מחרוזת המייצגת את המידע של המידע של הצומת עם המפתח המקסימלי.</li> <li>שיטת פעולה: <ul style="list-style-type: none"> <li>במידה והעץ לא ריק, על ידי שימוש במתודה empty, פונה לצומת בעל המפתח המקסימלי בעזרת השדה max, ומחזיר את המידע ממנו.</li> <li>אחרת, מחזיר null.</li> </ul> </li> <li>סיבוכיות: <math>O(1)</math>.</li> </ul>	
<b>Size</b>		<ul style="list-style-type: none"> <li>קלט: אין.</li> <li>פלט: מספר המייצג את כמות הצמתים בעץ = גודל העץ.</li> <li>שיטת פעולה: <ul style="list-style-type: none"> <li>מחזירה את הערך המספרי של גודל תת העץ (השדה subTreeSize) של העצם מסוג WAVLNode של השורש. <ul style="list-style-type: none"> <li>הפעולה מוגדרת היטב, מכיוון שביצירת הצומת האקסטרני, EXT, גודל העץ שלו מוגדר להיות 0.</li> </ul> </li> </ul> </li> <li>סיבוכיות: <math>O(1)</math>.</li> </ul>	
<b>getRoot</b>		<ul style="list-style-type: none"> <li>קלט: אין.</li> <li>פלט: WAVLNode המייצג את שורש העץ.</li> </ul>	

	<ul style="list-style-type: none"> <li>שיטת פעולה: <ul style="list-style-type: none"> <li>בודקת האם השורש מצביע לצומת האקסטרני, EXT, ומחזירה null אם כן. <ul style="list-style-type: none"> <li>יש לשים לב כי בעץ ריק השורש הוא הצומת האקסטרני.</li> </ul> </li> <li>אחרת, מחזירה את המצביע root באמצעות החזרת השדה התואם.</li> </ul> </li> <li>סיבוכיות: <math>O(1)</math>.</li> </ul>
<u>Select</u>	<ul style="list-style-type: none"> <li>קלט: מספר של פיו נחפש את האיבר ה-<math>i</math> בגודלו בעץ.</li> <li>פלט: מחרוזת המייצגת את המידע של הצומת ה-<math>i</math> בגודלו בעץ.</li> <li>שיטת פעולה: <ul style="list-style-type: none"> <li>בדיקות התחלה: <ul style="list-style-type: none"> <li>אם העץ ריק, על ידי המתודה empty, או שאינדקס הקלט גדול מכמות הצמתים בעץ, בעזרת המתודה size, מוחזר הערך null.</li> <li>אם מחפשים את האיבר הראשון, משמע הקטן ביותר, מחזיר את המידע מהצומת המינימלי, בעזרת המצביע min.</li> <li>אם מחפשים את האיבר האחרון, הבדיקה מתבצעת בעזרת המתודה size, מוחזר המידע מהצומת המקסימלי, בעזרת המצביע max.</li> </ul> </li> <li>במידה והמתודה לא נעצרה עד כה החיפוש נעשה בשיטה הבאה: <ul style="list-style-type: none"> <li>בכל איטרציה, נתייחס לתת העץ שאנו נמצאים בו, כאילו הוא עץ חדש, ללא זיכרון למה היה בעבר.</li> <li>מתחילים מהשורש.</li> <li>אם גודל תת העץ השמאלי, משמע כמות המפתחות שקטנים ממני, ועוד אחד (אני) הוא האינדקס אותו חיפשנו, אזי שאני הצומת ה-<math>i</math> בגודלו בעץ.</li> <li>אחרת, עלינו להמשיך לחפש על פי: <ul style="list-style-type: none"> <li>אם האינדקס גדול או שווה מגודל תת העץ השמאלי, אזי סימן שהוא נמצא בתת העץ השמאלי, ונחזור על הפעולה עד אשר נמצא אותו.</li> <li>אחרת, זה סימן שהוא נמצא בתת העץ הימני, אך הפעם, מכיוון ואנו מסתכלים על תת העץ הזה ללא זיכרון לעבר, עלינו למצוא כעת את האיבר ה-<math>i</math> פחות גודל העץ ועוד אחד, מכיוון ששללנו את יתר האיברים כבר, וכעת נחזור על הפעולה עד אשר נמצא אותו.</li> </ul> </li> </ul> </li> <li>לבסוף, נחזיר את המידע של הצומת עם המפתח בגודל ה-<math>i</math>.</li> </ul> </li> <li>סיבוכיות: <math>O(\log n)</math>.</li> </ul>
<u>Successor</u>	<ul style="list-style-type: none"> <li>קלט: עצם מסוג WAVLNode.</li> <li>פלט: WAVLNode המייצג את הצומת עם המפתח הבא בגודלו בעץ, אחרי צומת הקלט.</li> <li>שיטת פעולה: <ul style="list-style-type: none"> <li>במידה והעצם המתקבל הוא המקסימלי בעץ, על ידי שימוש במצביע, אזי אין מפתח גדול ממנו, ויוחזר null.</li> <li>אחרת, למדנו כי על מנת למצוא את המפתח הבא בגודלו עלינו לרדת לבן הימני, ואז שמאלה ברציפות ככל האפשר, והאחרון הוא היורש.</li> <li>או מנגד, במידה ואין בן ימני, כל עוד זה בן הימני של אביו, עלינו להמשיך לטפס עד אשר נמצא אב שאני אחיה הבן השמאלי שלו, והאחרון יהיה היורש.</li> <li>לבסוף, מחזירה את היורש שמצאנו.</li> </ul> </li> <li>סיבוכיות: <math>O(\log n)</math>.</li> </ul>
<u>Predecessor</u> גרסא רגילה	<ul style="list-style-type: none"> <li>קלט: עצם מסוג WAVLNode.</li> <li>פלט: WAVLNode המייצג את הצומת עם המפתח הקודם בגודלו בעץ, לפני צומת הקלט.</li> <li>שיטת פעולה: <ul style="list-style-type: none"> <li>קריאה לפונקציה predecessor בגסראתי הבולאנית, עם הoden שנקלט והערך הבוליאני false.</li> </ul> </li> <li>סיבוכיות: <math>O(\log n)</math>.</li> </ul>
<u>Predecessor</u> גרסא בולאנית	<ul style="list-style-type: none"> <li>קלט: עצם מסוג WAVLNode, ערך בוליאני המייצג האם קראנו למתודה מתוך deleteBinary או לא.</li> <li>פלט: WAVLNode המייצג את הצומת עם המפתח הקודם בגודלו בעץ, לפני צומת הקלט.</li> <li>שיטת פעולה: <ul style="list-style-type: none"> <li>במידה והעצם המתקבל הוא המינימלי בעץ, על ידי שימוש במצביע, אזי אין מפתח קטן ממנו, ויוחזר null.</li> <li>אחרת, למדנו כי על מנת למצוא את המפתח הקודם בגודלו עלינו לרדת לבן השמאלי, ואז ימינה ברציפות ככל האפשר, והאחרון הוא הקודם בגודלו. לאורך חיפוש זה, מתבצעת בדיקה האם המתודה נקראה מתוך deleteBinary. במידה וכן, עלינו להוריד את גודל תת העץ עבור צומת, מכיוון שאנו עתידים למחוק את הקודם בגודלו לאחר ההחלפה.</li> <li>או מנגד, במידה ואין בן שמאלי, כל עוד זה בן השמאלי של אביו, עלינו להמשיך לטפס עד אשר נמצא אב שמגיעים אליו מצד שמאל, והוא יהיה הקודם.</li> <li>לבסוף, מחזירה את המפתח של הקודם שמצאנו.</li> </ul> </li> <li>סיבוכיות: <math>O(\log n)</math>.</li> </ul>
<u>keysToArray</u>	<ul style="list-style-type: none"> <li>קלט: אין.</li> <li>פלט: מערך מספרים המכיל את כל המפתחות הקיימים בעץ.</li> <li>שיטת פעולה:</li> </ul>

<ul style="list-style-type: none"> <li>○ מקבלת את הערך של גודל העץ מהפונקציה <code>size()</code>.</li> <li>○ במידה וגודל העץ הוא 0, מוחזר מערך ריק.</li> <li>○ אחרת, נוצר מערך חדש בגודל העץ שקיבלנו קודם.</li> <li>○ כעת, מתבצעת לולאת <code>for</code> באורך גודל העץ של פעולות <code>successor</code> החל מהצומת המינימלי אשר מתקבל מהמצביע, ומכניסה למערך את המפתח של כל צומת.</li> <li>○ לבסוף מוחזר המערך המלא.</li> </ul>	
<ul style="list-style-type: none"> <li>● סיבוכיות: <math>O(n)</math>.</li> </ul>	<p><b><u>infoToArray</u></b></p> <ul style="list-style-type: none"> <li>● קלט: אין.</li> <li>● פלט: מערך מחרוזות המכיל את כל המידע הקיים בעץ.</li> <li>● שיטת פעולה:</li> <li>○ מקבלת את הערך של גודל העץ מהפונקציה <code>size()</code>.</li> <li>○ במידה וגודל העץ הוא 0, מוחזר מערך ריק.</li> <li>○ אחרת, נוצר מערך חדש בגודל העץ שקיבלנו קודם.</li> <li>○ כעת, מתבצעת לולאת <code>for</code> באורך גודל העץ של פעולות <code>successor</code> החל מהצומת המינימלי אשר מתקבל מהמצביע, ומכניסה למערך את המידע של כל צומת.</li> <li>○ לבסוף מוחזר המערך המלא.</li> </ul>
<ul style="list-style-type: none"> <li>● סיבוכיות: <math>O(n)</math>.</li> </ul>	<p><b><u>Rebalance</u></b></p> <ul style="list-style-type: none"> <li>● קלט: <code>WAVLNode</code> שהחל ממנו תתחיל הבדיקה לצורך באיזון.</li> <li>● פלט: מספר המייצג את כמות פעולות האיזון שנעשו.</li> <li>● שיטת פעולה:</li> <li>○ המתודה תבדוק עבור כל צומת אב החל מהקלט ועד לשורש את הצורך באיזון לפי הפרמטר המוחזר מהמתודה <code>deleteCases</code>:</li> <li>■ אם מוחזר 0: אין צורך לבצע אף פעולת איזון וממשיכים.</li> <li>■ אם מוחזר 1: מבצעים פעולת <code>demotion</code> לצומת וממשיכים.</li> <li>■ אחרת, אנו צריכים לעקוב אחרי איזה בן לא מהווה בעיה מבחינת הפרשי דרגות, ועל כן נשמור אותו בצד, בנוסף אליו, נשמור במשתנה <code>sonDir</code> מאיזה צד הוא הבן, ובמשתנה <code>direction</code> את הכיוון ההפוך (לטובת שימוש עתידי ברוטציות).</li> <li>■ אם מוחזר 2: מבצעים פעולת <code>double demotion</code> לצומת ולבן שלו ששמרנו בשלב הקודם, וממשיכים.</li> <li>■ אם מוחזר 3: מדובר בפעולת רוטציה אחת, ועל כן משתמשים במתודה <code>singleRotation</code> על הבן ששמרנו עם הכיוון ההפוך מהצד בו הוא נמצא, וממשיכים.</li> <li>■ אם מוחזר 4: מדובר בפעולת רוטציה כפולה, נפעיל את המתודה <code>singleRotation</code> פעמיים על פי הבן ששמרנו:</li> <li>● אם הוא ימני – נפעיל על בנו השמאלי של הבן, קודם רוטציה ימנית ואז שמאלה, וממשיכים.</li> <li>● אחרת, נפעיל על בנו הימני של הבן, קודם רוטציה שמאלה ואז ימנית, וממשיכים.</li> <li>○ לבסוף, מחזירים את כמות פעולות האיזון שנעשו.</li> </ul>
<ul style="list-style-type: none"> <li>● סיבוכיות: <math>O(\log n)</math>.</li> </ul>	<p><b><u>singleRotation</u></b></p> <ul style="list-style-type: none"> <li>● קלט: <code>WAVLNode</code> המייצג את הצומת שיהפוך להיות השורש של תת העץ לאחר הרוטציה, מחרוזות המייצגות את הכיוון אליו נעשה את הרוטציה.</li> <li>● פלט: <code>WAVLNode</code> המייצג את הצומת שהיה השורש של תת העץ לפני הרוטציה.</li> <li>● שיטת פעולה:</li> <li>○ בתחילה, אנו שומרים 3 משתנים: הצומת שקיבלנו, אביו וגודל תת העץ שהוא שורשו.</li> <li>○ כעת, אנו רוצים לחבר בין האבא של השורש המקורי, לשורש החדש. נעשית בדיקה האם המקורי הוא שורש העץ הגדול, ואם כן אזי שאין לו אבא, אז צריך לשנות את מצביע השורש של המחלקה להצביע, אחרת נעשית בדיקה האם ההחלפה נעשית מימין או משמאל וההשמה בהתאם.</li> <li>○ כעת, על סמך הכיוון שקיבלנו בקלט, מתבצעת סדרת פעולות של חיבור השורש הישן כבן של השורש החדש, ושל הבן הקודם של השורש החדש, לאביו החדש, שהוא השורש המקורי.</li> <li>○ כעת, מתבצעת בדיקה האם השורש המקורי הוא כעת עלה. אם כן, הוא מקבל ערכי גודל תת עץ ודרגה של עלה. אחרת, מוריד את דרגתו ב-1.</li> <li>○ לבסוף, מחזיר את השורש המקורי.</li> </ul>
<ul style="list-style-type: none"> <li>● סיבוכיות: <math>O(1)</math>.</li> </ul>	<p><b><u>Insert</u></b></p> <ul style="list-style-type: none"> <li>● קלט: <code>int</code> המייצג את המפתח ומחרוזות המייצג את המידע אשר רוצים להכניס לעץ יחד בתור איבר חדש.</li> <li>● פלט: מספר פעולות האיזון שהתרחשו בזמן הכנסת האיבר החדש לעץ.</li> <li>● שיטת פעולה:</li> <li>○ ראשית נבדקת האפשרות שהעץ ריק בעזרת המתודה <code>empty()</code> – אם כן, מייצרים <code>WAVLNode</code> חדש עם הערכים שנקלטו, ומשימים אותו בתור שדה השורש של העץ, כמו גם האיבר המקסימאלי ומינימאלי. מחזירים 0 שכן לא בוצעו פעולות איזון.</li> <li>○ במידה והעץ לא היה ריק, אנו עושים חיפוש בינארי על העץ, על מנת למצוא את המיקום בו צריך להכניס את האיבר החדש (אנו יודעים זאת על ידי הגעה לעלה אקסטירמי במקום הנדרש), ולשם מחברים איבר חדש עם הערכים שנקלטו (חיבור: הגדרה לאבא את האיבר כבן בכיוון הרלוונטי, ולאיבר את האבא בתור הורה). במידה ומגלים בחיפוש כי כבר קיים איבר עם המפתח שנקלט, מוחזר 1-.</li> </ul>

<ul style="list-style-type: none"> <li>במידה ואכן מצאנו כי ניתן להכניס את האיבר, עוברים בלולאה על כל הוריו של המקום הרצוי, ומגדילים להם את גודל subTreeSize.</li> <li>במידה וההורה היה אונארי, אין צורך בפעולות איזון, ומוחזר 0.</li> <li>נשמר משתנה countBalance שיספור את פעולות האיזון שמתבצעות. במידה וההורה היה עלה, נגדיל את rank של ההורה ב-1, ונבדוק את שלושת מצבי האיזון עבור ההורה של ההורה (parent) ועבור כל האיברים ממנו ועד השורש (או עד שמגיעים לרוטציה):</li> <li>מקרה 1: אם מגלים כי ההפרש הדרגות בין parent לבין בנו האחר הינו 1, ניתן להעלות בדרגה את parent, ולהעלות בבדיקה לאיבר הבא.</li> <li>מקרה 2: אם הפרשו של parent מבנו האחר הינו 2, נבדוק את הפרש הדרגות של בנו המקורי (זה שהעלנו במקור את דרגתו) מהנכד של parent אשר פונה "פנימה" - לכיוון בו נמצא parent. אם ההפרש הינו 2, מדובר במקרה 2, מבצעים רוטציה בודדת (אשר כוללת גם 2 שינויי דרגה), ומחזירים את סך פעולות האיזון.</li> <li>מקרה 3: אם ההפרש של הבן המקורי מהנכד שפונה "פנימה" הינו 1, יש צורך ברוטציה כפולה (שכוללת 4 שינויי דרגה) ומחזירים את סך פעולות האיזון.</li> <li>כל עוד לא מגיעים לפעולת רוטציה, או לשורש, ממשיכים בבדיקה זו עבור ההורה של ההורה הנוכחי.</li> <li>לבסוף – אם לא בוצעו כלל פעולות רוטציה, מחזירים את המשתנה שספר את מספר פעולות האיזון (אשר במקרה זה הינן רק פעולות העלאה בדרגה).</li> </ul>	
<ul style="list-style-type: none"> <li>סיבוכיות: <math>O(\log n)</math></li> </ul>	<p><b>Delete</b></p> <ul style="list-style-type: none"> <li>קלט: מספר המייצג את המפתח של הצומת אותו עלינו למחוק.</li> <li>פלט: מספר המייצג את כמות פעולות האיזון שביצענו.</li> <li>שיטת פעולה:</li> <li>תחילה, נבדק האם העץ ריק בשימוש המתודה empty, או האם המפתח לא קיים בעץ בשימוש המתודה search. אם אחד התנאים מתקיים אזי נחזיר 1.</li> <li>לאחר מכן, מתבצעת בדיקה האם הצומת היחיד בעץ הוא השורש, והוא גם המפתח הנדרש למחיקה, אם כן אנו מוחקים אותו על ידי הצבעה לצומת האקסטרני כעל השורש, ומחזירים כי ביצענו 0 פעולות איזון.</li> <li>כעת, אם לא נעצרנו עדיין, אנו מבצעים חיפוש בינארי עד אשר נמצא את הצומת הדרוש למחיקה, על פי המפתח.</li> <li>כאשר מצאנו את הצומת, אנו בודקים אם הוא הצומת המינימלי או המקסימלי בעץ. במידה וכן, אנו מעדכנים את המצביע המתאים לצומת הבא/הקודם.</li> <li>כעת, על פי סוג הצומת עלה/אונארי/בינארי, הוא נשלח למתודת המחיקה המתאימה.</li> <li>לאחר מכן, מבצעים פעולות rebalance במידת הצורך.</li> <li>לבסוף, מחזירים את כמות פעולות האיזון הכוללות שעשינו.</li> </ul>
<ul style="list-style-type: none"> <li>סיבוכיות: <math>O(\log n)</math></li> </ul>	<p><b>deleteLeaf</b></p> <ul style="list-style-type: none"> <li>קלט: WAVLNode המייצג את הצומת שעלינו למחוק.</li> <li>פלט: מספר המייצג את כמות פעולות האיזון שביצענו.</li> <li>שיטת פעולה:</li> <li>תחילה, אנו בודקים האם האב של הצומת הנמחק הוא אונארי, אם כן, סימן שיהפוך להיות עלה, ועל כן מקבל את השמה של כל הערכים המתאימים לעלה, ומחזיר 1, מכיוון ששינונו את דרגתו.</li> <li>אחרת, על סמך בדיקה של איזה צד הצומת הנמחק הוא בן של אביו, מתבצעת פעולה המחיקה.</li> <li>לאחר מחיקת העלה, מתקיימת בדיקה האם הבן השני של האבא הוא עלה, אם כן סימן שצריך רק לעשות בדיקות rebalance ולהחזיר את כמות פעולות האיזון שנעשו.</li> <li>אחרת, יהיה צורך ברוטציה, ולאחר מכן לבצע בדיקות rebalance, ולהחזיר את כמות פעולות האיזון שנעשו + 3, מכיוון שביצענו ברוטציה פעולות promotion, rotation, demotion.</li> </ul>
<ul style="list-style-type: none"> <li>סיבוכיות: <math>O(1)</math></li> </ul>	<p><b>deleteUnary</b></p> <ul style="list-style-type: none"> <li>קלט: WAVLNode המייצג את הצומת שעלינו למחוק.</li> <li>פלט: WAVLNode המייצג את אביו של הנמחק.</li> <li>שיטת פעולה:</li> <li>המתודה למעשה מקפיצה את האבא של הנמחק להכיר את הבן של הנמחק.</li> <li>בתחילה אנו מגלים מאיזה צד הצומת הנמחק הוא אונארי, ועל פי זה מבצעים את יתר הפעולות.</li> <li>נעשית בדיקה האם האב הוא עלה אקסטרני, ואם כן זה אומר שהנמחק הוא השורש, לכן צריך להשיב את הבן של הנמחק להיות השורש.</li> <li>אם הוא לא השורש, אז על פי איזה צד הצומת הנמחק הוא בן של אביו, מתבצעת ההתאמה בין אב הצומת לילד הצומת.</li> <li>לבסוף, מוחקים את הצומת, על ידי השמת אבא וילדים null, ומחזירים את צומת האב.</li> </ul>
	<p><b>deleteBinary</b></p> <ul style="list-style-type: none"> <li>קלט: WAVLNode המייצג את הצומת שעלינו למחוק.</li> <li>פלט: מספר המייצג את כמות פעולות האיזון שביצענו.</li> <li>שיטת פעולה:</li> <li>תחילה, אנו שומרים על מספר משתנים: <ul style="list-style-type: none"> <li>צומת ה-predecessor על ידי קריאה למתודה עם הערך הבוליאני true.</li> <li>מפתח ה-predecessor.</li> <li>מידע ה-predecessor.</li> </ul> </li> <li>כעת, אנו מבצעים פעולות מחיקה של ה-predecessor בהתאם להם הוא עלה או צומת אונארי.</li> </ul>

<ul style="list-style-type: none"> <li>○ לבסוף, מבצעים החלפת ערכים של הצומת שעלינו למחוק, על ידי החלפת המפתח והמידע שלו בזה של ה-predecessor שלו, ובכך למעשה מבצעים את מחיקתו.</li> <li>○ לבסוף מחזירים את כמות פעולות האיוון שעשינו.</li> </ul>	<ul style="list-style-type: none"> <li>• סיבוכיות: <math>O(\log n)</math>.</li> </ul>
<ul style="list-style-type: none"> <li>○ קלט: WAVLNode המייצג את הצומת עליו אנו מבצעים את הבדיקה איזה תצורת איוון נדרשת לאחר המחיקה שביצענו.</li> <li>• פלט: מספר המייצג את סוג תצורת המחיקה שעלינו לעשות.</li> <li>• שיטת פעולה:</li> <li>○ המתודה למעשה מחזירה איזה מבין ארבעת המקרים האפשריים של הפרת חוקי עץ WAVL לאחר מחיקה, אם בכלל יש הפרה, קיימת עבור הצומת אותו קיבלנו.</li> <li>○ תחילה, נבדקת האפשרות שאין הפרה כלל של החוקים, והמתודה תחזיר 0 המייצג שאין בעיה.</li> <li>○ לאחר מכן, בהתאם לאיזה צד ההפרה הקורית מתבצעות הבדיקות הנוספות.</li> <li>○ אם ההפרש לבן השני הוא 2, אזי ניתן לבצע פעולת demotion בלבד, לכן נחזיר 1.</li> <li>○ אם ההפרש לבן השני הוא 1, תחילה נשמור את הבן השני במשתנה ואז: <ul style="list-style-type: none"> <li>▪ אם ההפרש מהבן לשני בנוי הוא 2, אזי double demotion יהיה מספיק, ויוחזר הערך 2.</li> <li>▪ אם ההפרש מהבן ה"חיצוני" הוא 1, אזי נצטרך לבצע רוטציה, ויוחזר הערך 3.</li> <li>▪ אחרת, נצטרך לבצע רוטציה כפולה, ויוחזר הערך 4.</li> </ul> </li> </ul>	<p><b>deleteCases</b></p> <ul style="list-style-type: none"> <li>• סיבוכיות: <math>O(1)</math>.</li> </ul>

### מחלקה מקוננת – WAVLNode

שדות		
מכיל את מפתח ה-Node, כלומר הערך המספרי החי"ע אשר מעיד (וקובע) את מיקום האיבר בעץ, ולפיו נבצע את המתודות השונות של עץ WAVL. ערך זה מאוחסן ע"י קלט המשתמש, בעת יצירת איבר חדש.	int key	private
מכיל את הערך שהמשתמש הגדיר עבור הnode, בעת אתחולו. ערך זה אינו חח"ע.	String value	Private
קובע את דרגתו של הnode, בהתאם לכללי עץ WAVL, כלומר השדה תמיד יהיה בין 0 לבין פעמיים גובה העץ. בעת הכנסת איבר חדש לעץ, השדה של האיבר החדש מאוחסן להיות 0, שכן הוא יהיה עלה (אשר דרגתו תמיד 0). שדה זה משמש את פעולות ההכנסה ומחיקה של העץ, על מנת לקבוע את הסיטואציה, וכן השדה מעודכן על פי הסיטואציה המתאימה.	int rank	Public
מפנה אל הnode אשר מהווה הורה של הnode הנוכחי. השדה מאוחסן באמצעות עם בניית האיבר, ע"י קלט רלוונטי. נשים לב כי ההורה של שורש העץ הינו null.	WAVLNode parent	Public
מפנה אל בנו השמאלי של הnode (אשר גם הוא node). בזמן בניית הnode, השדה מאוחסן להיות עלה אקסטרני EXT, שכן האיבר המדובר הינו עלה. בדומה לשדה left, אך מצביע על הבן הימני.	WAVLNode left	Public
שומר את גודל תת העץ (מספר האיברים) הקיים מתחת לאיבר הספציפי, כולל הוא עצמו. מתוחזק בזמן פעולות ההכנסה ומחיקה של העץ.	WAVLNode right	Public
	int subTreeSize	Public
מתודות		
<p><b>בנאי</b></p> <ul style="list-style-type: none"> <li>• קלט: מפתח מסוג int, ערך מסוג string, איבר מסוג WAVLNode</li> <li>• שיטת פעולה: הבנאי נקרא בכל פעם שמאוחסן node חדש. המפתח והערך מושמים בתור מפתח וער האיבר.</li> <li>• WAVLNode parent מושם בתור ההפניה להורה האיבר. כמו כן, הrank של האיבר מאוחסן להיות 0 (בהתאם לכך שהינו עלה), subTreeSize מאוחסן להיות 1 ושדות הבנים מאוחסנים להיות EXT. נשים לב כי עבור עלה שהינו EXT ישנו אתחול מיוחד (המזוהה באמצעותה כנסת מפתח -1).</li> <li>• סיבוכיות: <math>O(1)</math>.</li> </ul>		
<p><b>gets</b></p> <ul style="list-style-type: none"> <li>• לכל שדה ישנה מתודת get, אשר מחזירה את ערך השדה בסיבוכיות <math>O(1)</math>. נשים לב כי תחזוק השגות באופן חוקי קורה במהלך ביצוע מתודות WAVLTree.</li> <li>• קלט: אין</li> <li>• פלט: השדה הרלוונטי לאותה מתודה</li> <li>• שיטת פעולה: מחזיר את השדה השמור</li> <li>• סיבוכיות: <math>O(1)</math></li> </ul>		
<p><b>isInnerNode</b></p> <ul style="list-style-type: none"> <li>• קלט: אין (מתודת מופע על איבר ספציפי)</li> <li>• פלט: ערך בולאני</li> <li>• שיטת פעולה: המתודה בודקת אם שדה ה-rank האיבר גדול או שווה לאפס. אם כן, סימן שמדובר באיבר פנימי, והמתודה מחזירה true. אחרת, סימן שמדובר בעלה אקסטרני, והמתודה מחזירה false.</li> <li>• סיבוכיות: <math>O(1)</math></li> </ul>		

מדידות:

מספר פעולות האיזון המקסימלי לפעולות delete	מספר פעולות האיזון לממוצע לפעולות delete	מספר פעולות האיזון המקסימלי לפעולות insert	מספר פעולות האיזון הממוצע לפעולות insert	מספר פעולות	מספר סידורי
9	2.0339	15	3.4042	10,000	1
10	2.02275	18	3.38685	20,000	2
10	2.02296	18	3.43103	30,000	3
11	2.020375	19	3.41205	40,000	4
10	2.03228	18	3.40918	50,000	5
11	2.025916	18	3.41686	60,000	6
11	2.03111	18	3.40737	70,000	7
12	2.033925	19	3.41678	80,000	8
12	2.031966	19	3.40765	90,000	9
12	2.04219	19	3.40261	100,00	10

• ציפיות על בסיס רקע תיאורטי:

- מספר פעולות האיזון הממוצע לפעולות insert ולפעולות delete צפוי להיות מספר קבוע -  $O(1)$  ללא תלות בגודל העץ או בגובהו.
- מספר פעולות האיזון המקסימלי לפעולות insert או delete צפוי להיות מסדר הגודל של גובה העץ-שכידוע חסום מלמעלה ע"י  $2 \log n$  – כאשר  $n$  הוא מספר האיברים בעץ. לכן, נצפה שמספר זה יהיה מסדר הגודל של  $\log(n)$ , ויגדל במתינות עם גידול כמות האיברים בעץ.

• תוצאות המדידות:

- מספר פעולות האיזון הממוצע לפעולות insert או delete בכל אחת מהמדידות היה **קבוע** (עם סטיות זניחות), ולא השתנה כפונקציה של מספר האיברים בעץ. אם כך, ניתן להסיק שתוצאות אלה עולות בקנה אחד עם ניתוח ה-amortized התיאורטי של מספר פעולות האיזון עבור פעולת הכנסה ומחיקה שהתבצע בהרצאה.
- מספר פעולות האיזון המקסימלי הן עבור פעולות insert בודדת והן עבור פעולות delete בודדת היה מסדר הגודל של  $\log n$  – כאשר  $n$  הוא מספר האיברים בעץ. מספר זה גדל במתינות ככל שהגדלנו את כמות האיברים בעץ. ניתן להסיק כי גם תוצאות אלו תואמות את הניתוח התיאורטי שבוצע בכיתה.