מבני נתונים 1 – תרגיל בית רטוב 1 – אביב תשפ"א

שמות הסטודנטים: ליאור בר, ליאור רסקין

מס' ת"ז (בהתאמה): 209163559, 316216977

שם הקורס: מבני נתונים 1

מס' הקורס: 234218

סוג התרגיל: תרגיל בית רטוב 1 (חלק יבש)

תאריך הגשה: 24/05/2021

הערה: מלבד עמוד השער, חלק זה מכיל 8 עמודים המסכמים את החלק היבש של תרגיל בית 1 הרטוב בהתאם להנחיות התרגיל.

תיאור מבני הנתונים:

ראשית, נציין שעיקר ניהול הנתונים שלנו ייעשה בעזרת מחלקת עץ AVL גנרית שיכולה לקבל ולנהל בתוכה איברים מכל סוג שהוא. על האיבר שהעץ מקבל, להכיל את המפתח בתוכו (כלומר, אופרטור " > ").

מחלקות העזר:

- **Model:** מחלקה המייצגת דגם של רכב.

המחלקה תגדיר ותאפשר גישה לנתונים הבאים של הדגם: מס' זיהוי סוג הרכב של הדגם, מס' זיהוי הדגם, ציון הדגם, מס' המכירות מהדגם, מס' התלונות שהרכב קיבל.

המחלקה תאפשר עדכון של מכירה ותלונה בהתאם למפתח הניקוד שהוגדר בתרגיל.

המחלקה תאפשר השוואה בין דגמים, באמצעות אופרטורים, והתאם למפתח העדיפויות שהוגדר בתרגיל: ציון, מזהה סוג רכב, מזהה דגם רכב.

- **ModelBySale:** מחלקה המייצגת דגם שמיועד להשוואה בהתאם למס' המכירות ממנו.

המחלקה תגדיר ותאפשר גישה לנתונים הבאים של הדגם: מס' זיהוי סוג הרכב של הדגם, מס' זיהוי הדגם, מס' המכירות מהדגם.

המחלקה תאפשר השוואה בין דגמים, באמצעות אופרטורים, ובהתאם למפתח שנקבע בתרגיל: מס' המכירות הגבוה ביותר, מזהה סוג הרכב הקטן ביותר, מזהה דגם הרכב הקטן ביותר.

- **ModelsArray:** מחלקה המייצגת מערך של דגמים שמשויך לסוג רכב במערכת.

המחלקה תגדיר ותאפשר גישה לנתונים הבאים: סוג הרכב המגדיר את המערך, מס' הדגמים השמורים במערך, עותק של הדגם הנמכר ביותר, מצביע למערך הדגמים.

המחלקה תאפשר לעדכן את הדגם הנמכר ביותר תחת מזהה סוג הרכב על בסיס המפתח שהוגדר בתרגיל: מס' המכירות הגבוה ביותר, מזהה סוג הרכב הקטן ביותר, מזהה דגם הרכב הקטן ביותר.

המחלקה תאפשר השוואה בין מערכי הדגמים באמצעות אופרטורים ובהתאם למזהה סוג הרכב המגדיר כל מערך.

- **TypeTree:** מחלקה המייצגת תת-עץ של דגמים (אובייקט Model) עם ציון אפס.

המחלקה תגדיר ותאפשר גישה לנתונים הבאים של תת-העץ: מזהה סוג הרכב שמשויך לכל הדגמים בתת-העץ, מס' הדגמים בתת-העץ, מצביע לתת-העץ.

המחלקה תאפשר השוואה בין תתי-העצים באמצעות אופרטורים ובהתאם למזהה סוג הרכב המגדיר כל תת-עץ.

- **AvlTree:** מחלקה המייצגת עץ AVL גנרי (ע"י תבנית). המחלקה תגדיר את הנתונים הבאים:
  - ערך גנרי של כל צומת (מפתח הסידור בעץ יהיה האופרטור שהוגדר על הערך).
  - מצביע לבן ימני.
  - מצביע לבן שמאלי.
  - צומת דמה המאפשר גישה ישירות לשורש העץ, שבו מאותחלים ערכי זבל (רלוונטי רק עבור צומת אחד בעץ).
  - מצביע בשם next, שרלוונטי רק כדי להתקדם מאיבר הדמה בראש העץ, לעץ עצמו.

- מצביע לאב של הצומת.
- מצביע לאיבר עם הערך המינימלי בעץ (מצביע כזה קיים רק בשורש העץ).
- מצביע לאיבר עם הערך המקסימלי בעץ (מצביע כזה קיים רק בשורש העץ).
- גובה העץ.

ברמה העקרונית, המחלקה תאפשר למשתמש את הפעולות הבאות:

- בניית עץ AVL גנרי והריסתו בהתאם לסיבוכיות המוכרת מההרצאה.
- בניית עץ AVL עם n ערכים ממוינים מראש שיתקבלו בתור מערך ממיין ביעילות של O(n). אופן הפעולה יפורט בהמשך.
- פעולות חיפוש, הכנסה והוצאה כפי שנלמדו בהרצאה (עם גלגולים) ובהתאם לסיבוכיות הלוגריתמית הנדרשת.
- קבלת גישה להורה/בנים של צומת בעץ (גישה ישירה לשדות של העץ בסיבוכיות של o(1)).
- פעולות המאפשרות קבלת גישה למצביע של ערך בעץ וקבלת גישה לערך של צומת נתון בעץ, ע"י פעולת החיפוש (סיבוכיות לוגריתמית).
- פעולות המאפשרות הכנסת ערך גנרי עבור בן שמאלי/ימני של צומת נתון בעץ, ע"י בניית איבר חדש בעץ בסיבוכיות של o(1).
- פעולות פנימיות שמוצאות את המצביע לאיבר המינימלי/מקסימלי בעץ, ע"י רקורסיה שמתבצעת על הענף השמאלי/ימני (בהתאמה) בסיבוכיות לוגריתמית, ובהתאם למפתח שהוגדר על הערך הגנרי. (הפעולות מתבצעות מיד לאחר כל הכנסה או הוצאה של איבר וכך השדות של האיבר המינמאלי והמקסימלי מתעדכנים בכל הכנסה והוצאה ושומרים על היעילות של log(n))
- פעולות שמחזירות את הערך של האיבר המינימלי/מקסימלי בעץ בהתאם למפתח שהוגדר על הערך הגנרי (גישה ישירה לשדות של העץ בסיבוכיות של o(1).
- בדיקה האם העץ ריק (גישה ישירה לשדה בסיבוכיות o(1).

המחלקה הראשית – תיאור כללי של מבני הנתונים במחלקה:

- **Types_tree:** עץ AVL שמייצג את כל סוגי הרכבים במערכת (לפי סדר מזהה סוג הרכב). הערך בכל צומת בעץ יהיה אובייקט של המחלקה ModelsArray. כל צומת בעץ ייצג סוג רכב וישמור את מערך הדגמים המשויך לו.
- **Models_tree:** עץ AVL שמייצג את כל הדגמים במערכת שהציון שלהם שונה מאפס (לפי המפתח שהוגדר בתרגיל על השוואת דגמים בהתאם לציון). הערך בכל צומת בעץ יהיה אובייקט של המחלקה Model. כל צומת בעץ ייצג דגם במערכת (ללא קשר לסוג הרכב אליו הוא משויך).
- **Models_tree_by_sale:** עץ AVL שמייצג דגמים המהווים את הדגם הנמכר ביותר של כל סוג הרכב שלהם במערכת (לפי המפתח שהוגדר בתרגיל על השוואת דגמים לפי מס' מכירות). הערך בכל צומת בעץ יהיה אובייקט של המחלקה ModelBySale. כל צומת בעץ מייצג את הדגם הנמכר ביותר של סוג רכב שלו כך שמס' האיברים בעץ זהה למס' האיברים בעץ Types_tree.
- **Zero_tree:** עץ AVL שמייצג את כל הדגמים במערכת שהציון שלהם הוא אפס (להלן "עץ האפסים"). הערך בכל צומת יהיה תת-עץ שיהווה אובייקט של המחלקה TypeTree . העץ הראשי (החיצוני) יהיה מסודר על פי מזהה סוג רכב וכל צומת בתת-העץ ייצג דגם המשויך לסוג הרכב הנ"ל (כלומר אובייקט Model).
- בנוסף נשמור במחלקה הראשית משתנה בוליאני ומונה הדפסות שמסייעים בפעולת GetWorstModels.

סיבוכיות המקום הכוללת של המבנים והפעולות:

במקרה הגרוע, כלומר בהנחה והעצים מלאים, אנו מאחסנים איברים בארבעת העצים כך שכל האיברים שמשוחררים בפעולה ()Quit למעשה צריכים להילקח בחשבון כחלק מסיבוכיות המקום של המבנים והפעולות.
מתקיים בעבור ארבעת העצים ומערך הדגמים:

- **types_tree:** איברי העץ כאמור מהווים את סוגי הרכב, כלומר **n** איברים.
- **n** מערכים אשר **כל איבריהם יחדיו** מייצגים את **m** הדגמים במערכת.
- **models_tree:** איברי העץ מהווים לכל היותר את מס' הדגמים הכולל במערכת, כלומר לכל היותר **m** איברים (כאשר **m** הוא המשתנה שהוגדר בהוראות בתרגיל בפעולה זאת).
- **models_tree_by_sale:** איברי העץ מהווים דגם עבור כל סוג רכב, כלומר סה"כ **n** איברים.
- **zero_tree:** איברי העץ מהווים השלמה של **models_tree** למס' הדגמים במערכת, ובכל מקרה לא עוברים את מס' הדגמים במערכת. כלומר, לכל היותר **m** איברים.

לפיכך סה"כ סיבוכיות המקום של המבנים תהיה **O(n+m)** כנדרש, זאת כאשר **n** מייצג את מס' סוגי הרכב במערכת ו-**m** מייצג את מס' הדגמים הכולל במערכת.

המחלקה הראשית – הפעולות הממומשות בתרגיל וסיבוכיות הזמן:

הערה: השימוש במשתנים **n,m,M** הוא בהתאם להתייחסות אליהם בהוראות התרגיל בכל פעולה.

- **Init():**

הפעולה ממומשת באמצעות פעולת האתחול (הבנאי) של המחלקה הראשית. הפעולה מאתחלת את ארבעת העצים ללא צמתים (מלבד צומת הדמה המייצג עץ ריק). מדובר באתחול מצביעים ושדות פרימיטיביים ולפיכך סיבוכיות הזמן של הפעולה היא **O(1)** כנדרש.

- **Quit():**

הפעולה ממומשת באמצעות פעולת השחרור (ההורס) של המחלקה הראשית. החלק שמומש על ידנו בהורס מבצע סריקה בעץ **zero_tree** ומבצע מחיקה של כל הקצאת איבר בו. כאמור, מס' האיברים בו הוא לכל היותר מס' סוגי הרכב בעץ, שמסומן ע"י **n** בסעיף זה בהוראות בתרגיל. בנוסף, ישנה קריאה להורסים של שדות המחלקה הראשית. מעבר לשדות הפרימיטיביים, יתבצעו קריאות להורסים הסטנדרטיים של ארבעת העצים. ההורסים יפעלו בדומה לעצי AVL שהוצגו בהרצאה. נתמקד בארבעת העצים:

  - **types_tree:** איברי העץ כאמור מהווים את סוגי הרכב, כלומר **n** איברים.
  - **n** מערכים אשר **כל איבריהם יחדיו** מייצגים את **m** הדגמים במערכת.
  - **models_tree:** איברי העץ מהווים לכל היותר את מס' הדגמים הכולל במערכת, כלומר לכל היותר **m** איברים (כאשר **m** הוא המשתנה שהוגדר בהוראות התרגיל בפעולה זאת).
  - **models_tree_by_sale:** איברי העץ מהווים דגם עבור כל סוג רכב, כלומר סה"כ **n** איברים.
  - **zero_tree:** איברי העץ מהווים השלמה של **models_tree** למס' הדגמים במערכת, ובכל מקרה לא עוברים את מס' הדגמים במערכת. כלומר, לכל היותר **m** איברים.

סה"כ ישנה סריקה על **3n+3m** איברים לכל היותר כאשר כל סריקה מבצעת שחרור ב-**O(1)**.
לפיכך סה"כ סיבוכיות הזמן של הפעולה תהיה **O(n+m)** כנדרש, במקרה הגרוע.

- **AddCarType():**

הפעולה מבצעת הקצאה של אובייקט מסוג **ModelsArray** עם הפרמטרים שהתקבלו. בבנאי של אובייקט זה מתבצעת הקצאת מערך של אובייקטים מסוג **Model**. לאחר מכן בבנאי אנו מבצעים סריקה על המערך (כלומר על **m** איברים) ומבצעים הקצאה של אובייקט **Model** לכל תא במערך. ההקצאות עצמן מתבצעת ב-**O(1)**.
לאחר מכן אנו יוצרים תת-עץ שמכיל אובייקטים מסוג **Model** שאותם אנחנו מקבלים ממוינים מהמערך של הדגמים שזה עתה יצרנו. (נזכור שלכולם אפס נקודות ולכולם אותו ID ולכן המיון

שלהם הוא לפי מספר הדגם בלבד). לכן, נוכל ליצור עץ AVL של דגמים כאלו בעזרת הבנאי השני שבנינו בעץ, שמקבל m דגמים ממוינים, ובונה עץ ביעילות של **O(m)**.

הדרך שבה עובדת הפעולה היא פשוטה: בכל פעם, היא תכניס את האיבר האמצעי במערך שהיא קיבלה לראש העץ, ולאחר מכן באופן רקורסיבי תכניס לבן הימני את האיבר האמצעי מתוך החצי הימני של המערך, ותכניס לבן השמאלי את האיבר האמצעי מתוך החצי השמאלי של המערך. ככה, בכל פעם אנו עוברים ומתקדמים באופן רקורסיבי צומת אחרי צומת, עד שנבנה בדיוק m צמתים. ולכן, היעילות בפעולה זו היא **O(m)**.

לאחר בניית תת העץ הנ"ל, נקצה אובייקט חדש מסוג **TypeTree** ונשייך אליו את תת העץ שזה עתה בנינו על ידי העברת כתובת המצביע של העץ אליו. לאחר מכן נכניס לעץ **zero_tree** בסיבוכיות לוגריתמית שהוכחה בהרצאה את האובייקט **TypeTree** שזה עתה בנינו על מנת להכניס אל העץ, ששומר על כל הדגמים שהציון שלהם אפס, את הדגמים שהתווספו עכשיו.

נזכיר כי כל הדגמים החדשים עם ציון אפס. **עד כאן, אנו בסיבוכיות של O(log(n) + m) מפעולות אלו שכן בניית העץ הייתה O(m) והכנסת תת העץ כולו לעץ "האפסים" הייתה ביעילות log(n) כפי שנלמד בהרצאה.**

לסיום אנו מקצים אובייקט **ModelBySale** בסיבוכיות של **O(1)** ומבצעים הכנסה שלו בסיבוכיות לוגריתמית לעץ עם **n** איברים (בכל צומת איבר עם מספר מכירות מקסימאלי מאותו סוג ID). מדובר באובייקט המייצג את הדגמים החדשים בהתאם למפתח הנתון לדגם הנמכר ביותר.

סה"כ ביצענו מס' קבוע של הכנסות בסיבוכיות לוגריתמית ופעולת סריקה על המערך. לפיכך, סיבוכיות הזמן של פעולה זאת תהיה **O(log(n) + m)** כנדרש.

- **<u>RemoveCarType():</u>**

באופן דומה לפעולה הקודמת, נתחיל בהקצאת אובייקט **ModelsArray** עם סריקה של **m** איברים. נמשיך בביצוע פעולת חיפוש עם סיבוכיות לוגריתמית על עץ עם **n** איברים במטרה למצוא את סוג הרכב הנתון בפעולה בעץ סוגי הרכב.

בהנחה והקיים רכב עם המזהה הנתון נרצה תחילה למחוק את הדגמים ששייכים לו מעץ האפסים. נקצה אובייקט ונחפש אותו בעץ עם **n** איברים. אם לא נמצא העץ, סימן שאין דגמים עם ציון אפס תחת המזהה הנתון ולפיכך ניתן להמשיך. אם נמצא, יש לשחרר את כלל הדגמים הללו ולהוציא את תת-העץ הרלוונטי מעץ האפסים. פעולת שחרור הדגמים מתבצעת על **m** איברים בעץ והסרת תת-העץ מתבצעת בסיבוכיות לוגריתמית על עץ עם **n** איברים.

נמשיך בהסרת הצומת המייצג בעץ **models_tree_by_sale** ע"י יצירת אובייקט ב-**O(1)** והסרתו מעץ עם **n** איברים.

לאחר מכן נרצה להסיר מהעץ **models_tree** את כל הדגמים השייכים למזהה הנתון שהציון שלהם שונה מאפס ולפיכך נמצאים בעץ זה. נרוץ בלולאה של **m** הדגמים המיועדים למחיקה ונבדוק אם הציון שלהם שונה מאפס. במקרה הגרוע (כלומר בהנחה שהתשובה חיובית), נסיר

אותם מעץ עם **M** איברים בסיבוכיות לוגריתמית. כלומר, סה"כ נפעיל את סיבוכיות זאת **m** פעמים לכל היותר במקרה הגרוע.

לסיום, נסיר את האובייקט **ModelsArray** שיצרנו מהעץ **types_tree** כך שמרגע זה לא קיים עוד מזהה סוג הרכב שניתן הפעולה. ההסרה תתבצע על עץ של **n** צמתים בסיבוכיות לוגריתמית.

סה"כ אנו מבצעים מס' קבוע של סריקות על **m** איברים, מס' קבוע של סריקות על **log(n)** איברים, **m** סריקות על **log(M)** איברים (כלומר סה"כ **m*log(M)** איברים).

נשים לב כי ניתן להניח ש-**log(M)** אינו קבוע ובפרט גדול מ-1 ולכן סיבוכיות הזמן הכוללת של פעולה זאת תהיה **O(log(n) + m*log(M))** כנדרש.

- **<u>SellCar():</u>**

נתחיל מכל בדיקות הקלט בשאלה, ונחזיר "קלט שגוי" אם הקלט לא יעמוד בתנאים של השאלה.

אם כן, נבדוק אם הדגם שאני רוצים למכור בכלל קיים במערכת ע"י חיפוש האיבר בעץ **types_tree** בשיטה שלמדנו בהרצאה. (קודם נקצה עם המחלקה **ModelsArray** איבר דמה, ואז נחפש אותו שכן המפתח אצלנו מוטמע בתוך המחלקות עצמן שבנינו). אם לא קיים במערכת, נחזיר "כישלון". במידה והוא קיים בעץ, תחילה נבדוק כמה נקודות יש לדגם עכשיו ונשמור מספר זה. לאחר מכן, נעדכן במערך את הנקודות וכמות המכירות של הדגם הנוכחי וגם ניצור עותק חדש שלו שנשתמש בו בהמשך.

לאחר מכן נעדכן במידת הצורך את האיבר עם הכי הרבה מכירות אם כעת איבר זה עבר בכמות המכירות שלו את האיבר המקסימלי הקודם בתוך כל האיברים במערך תחת ה-ID שקיבלנו. כל זה יהיה ביעילות **log(n)** כפי שנלמד בהרצאה.

אם אכן האיבר עם מספר המכירות המקסימלי השתנה, נוציא מעץ **models_tree_by_sale** את האיבר הקודם בעל אותה כמות המכירות בעזרת השיטה שנלמדה בהרצאה להוצאת איבר מעץ AVL ונכניס במקומו את האיבר החדש עם כמות המכירות החדשה גם בשיטה שלמדנו בהרצאה. וגם זה, יהיה ביעילות לוגריתמית שכן נעשה מספר קבוע של פעולות הכנסה והוצאה שכל אחת מהן לוגריתמית על **n** האיברים שבעץ.

לאחר מכן נבדוק כמה נקודות היו לעץ לפני ששינינו אותן בערך ששמרנו קודם. אם היו אפס נקודות, נוציא את האיבר מעץ האפסים, על ידי מציאת הצומת הנכון של תת העץ של הדגמים בעץ האפסים ביעילות לוגריתמית, ואז נוציא ממנו את הדגם מתת העץ שמצאנו גם כן ביעילות לוגריתמית כמו שלמדנו.

לאחר מכן, ניקח את הדגם החדש שיצרנו מקודם ונכניס אותו הפעם לעץ **models_tree** במקום. במידה והציון לפני השינוי לא היה אפס, אז נפעל הפוך. תחילה נוציא אותו מהעץ **models_tree,** ונבדוק מה הציון של הדגם המעודכן החדש. אם הוא אפס, אז נכניס את הדגם לעץ האפסים בדיוק כפי שתואר קודם (נמצא את הצומת של תת עץ הדגמים בעץ האפסים לפי האלגוריתם למציאת צומת בעץ, ואז נכניס את הדגם לתת העץ שמצאנו). אם הוא שונה מאפס, נכניס את הדגם המעודכן בחזרה לעץ **models_tree.**

כל הפעולות שאנחנו עושים כאן הן קבועות ביעילות של O(1) מלבד הכנסה, הוצאה, ומציאת איבר בעץ AVL שלהן יעילות לוגריתמית. וכמות הפעולות הללו היא קבועה. לכן במקרה הגרוע ביותר, נבצע פעולות אלה על העצים models_tree ו- types_tree שבהם M ו- n איברים בהתאמה במקרה הגרוע ביותר, ולכן היעילות תהיה O(log(n) + log(m)).

- **<u>MakeComplaint():</u>**

הפעולה עובדת כמעט זהה לחלוטין לפעולה הקודמת, מלבד העובדה שאין התייחסות לכמות המכירות אלא רק לניקוד, שמחושב בהתאם למפתח שהוגדר בתרגיל, ובפעולה זו אין התייחסות לעץ Models_tree_by_sale. מעבר לזה, שאר התהליך נעשה באותו האופן, ועל כן גם היעילות תישאר זהה: O(log(n) + log(m)).

- **<u>GetBestSellerModelByType():</u>**

במידה וה- ID הוא אפס, ניגש ישירות לשדה שמייצג את הערך המקסימלי בעץ ה AVL **Models_tree_by_sale** ששמרנו שתמיד מתעדכן בעץ בעת הכנסה או הוצאה. השדה יכיל כבר את הדגם עם מספר המכירות המקסימלי מבין כל הדגמים השונים. הפעולה תתבצע ב- **O(1).**
במידה וה- ID הוא לא אפס, תחילה נמצא את הצומת עם אותו ה- ID בעץ **ModelsArray,** ובצומת זה במחלקה ניגש לשדה המיוחד ששמרנו שמכיל את הדגם עם מספר המכירות הרב ביותר תחת ה-ID הנ"ל. מציאת הצומת זה כמובן גישה ביעילות של **log(n)** , ושאר הפעולות קבועות. לכן, במקרה הראשון היעילות הוא **O(1)** ובמקרה השני, **O(log(n)).**
וכמובן במקרה ולא נמצא האיבר בעץ שלנו, נחזיר "כישלון".

- **<u>GetWorstModels():</u>**

בפעולה זו בנינו מספר פעולות עזר שניעזר בהן:

- <u>checkForZeroParent()</u> – הפעולה תבדוק האם הצומת הנוכחי קטן מאפס וההורה שלו גדול מאפס. הסיבוכיות היא **O(1)**, שכן מדובר בכמה תנאים שבודקים במספר קבוע של פעולות.
- <u>checkForZeroSubTreeRoot()</u> – הפעולה תבדוק האם האב הקדמון של בן שאין לו עוד בנים שמאליים הוא שלילי והבן עצמו חיובי. הסיבוכיות היא **O(1)**, שכן מדובר בכמה תנאים בוליאניים פשוטים שבודקים במספר קבוע של פעולות.
- <u>goThroughModelsTreeInOrder()</u> – פעולה שעוברת על תת עץ שהיא מקבלת בשיטת InOrder ומכניסה כל איבר בתורו למערך שמקבלים בפעולה GetWorstModels().
- <u>goThroughModelsTree()</u> - פעולה שמקבלת צומת בעץ(במקרה שלנו את הצומת המינימאלי בעץ), מכניסה אותו למערך, לאחר מכן נעזרת בפעולה הקודמת למעלה כדי לעבור על תת העץ הימני מהצומת הנוכחי בשיטת InOrder ולהכניס בו את כל הצמתים במערך. לאחר מכן, הפעולה עולה להורה של הצומת הנוכחי, וחוזרת על התהליך שוב על כל

הענף השמאלי של העץ עד שהיא תגיע לשורש או עד ש- **printed_counter** יהיה שווה למספר הצמתים שהתבקשנו לעבור עליהם (כל זה נעשה בלולאה).

- goThroughZeroTree() – פעולה שדומה ל- goThroughModelsTree(), רק מתבצעת על עץ האפסים. מוצאת את הצומת המינימאלי שלו, ועוברת על העץ בדיוק על פי אותם העקרונות.
- goThroughZeroTreeInOrder() - פעולה שדומה ל- goThroughModelsTreeInOrder(), רק מתבצעת על עץ האפסים.

הרעיון הכללי של הפעולה: בכל פעם שנגיע לצומת ונעביר אותו למערך, נוסיף 1 לשדה printed_counter כדי לעקוב אחרי כמות האיברים שאנחנו מכניסים למערך וכדי שלא נחרוג מכמות האיברים שהתבקשנו להחזיר בתחילת הפעולה.

בפעולה זו, נמצא תחילה את הצומת המינימאלי ששמור לנו בעץ **Models_tree**. אם הציון שלו חיובי, נעבור קודם על עץ האפסים עם הפעולה goThroughZeroTree(). ולאחר מכן, על העץ הרגיל על ידי הפעולה goThroughModelsTree(). במידה והציון שלילי, נתחיל בהדפסת עץ הדגמים על ידי הפעולה goThroughModelsTree(). נשים לב, שבפעולה זו אנחנו גם נעזרים בשתי הפעולות checkForZeroSubTreeRoot() ו- checkForZeroParent() שנועדו לבדוק מקרים בהם עברנו על צומת שלילי, והצומת הבא שאנחנו עוברים אליו הוא חיובי. שכן אם זה קורה, תחילה יש לעבור על כל עץ האפסים שבו לכל הדגמים ציון אפס ורק אז לחזור לנקודה שבה עצרנו ולהמשיך להדפיס את עץ הדגמים הרגיל.

יש שני מקרים שונים שצריך לבדוק שבהם יש מעבר שכזה בין צומת שלילי לצומת חיובי. הראשון הוא מצב שבו החזרנו עכשיו צומת שלילי, ואנו עולים במעלה העץ ולכן נבדוק האם ההורה הוא חיובי. ולכך אחראית הפעולה checkForZeroParent(). מצב נוסף הוא, שיש אב קדמון שלילי שכבר הדפסנו בזמן שעלינו במעלה העץ, אבל יש לאב קדמון זה בן ימני חיובי שאין לו עוד בנים שמאליים שליליים. ולכן, נרצה גם לבדוק מצב כזה ובו לפני כן לעבור על עץ האפסים לפני שממשיכים עם עץ הדגמים הרגיל. ולכן, לצורך בדיקה זו יש את הפעולה checkForZeroSubTreeRoot().

באופן כללי, מה שאנחנו עושים זה לעבור על הענף השמאלי מהבן הקטן ביותר של העץ ולעלות במעלה העץ, כשבכל פעם אנחנו עוברים בשיטת ה- InOrder על תת-העץ הימני של הצומת שבה אחנו נמצאים. כל זה יחדיו הוא ביעילות של **O(m)** שכן המעבר דומה מאוד לשיטת ה-InOrder רק שהוא מתחיל מהצומת המינימלי בעץ ישירות ולא מראש העץ, ולכן במקום לעבור על כל הצמתים בעץ, אנחנו עוברים על מספר צמתים ידוע מראש בהתאם ל- numOfModels שנקבל בפעולה.