מבני נתונים – תרגיל 2 (מעשי)

הפעלת אופרטורים על אוסף מספרים, עבודה עם עצי סגמנטים

תאריך פרסום: 11.04.2023

תאריך הגשה: 2.5.2023 ,23:59

מתרגל אחראי: ענבל רושנסקי

הנחיות:

- יש לקרוא הנחיות לגבי הגשת העבודה באתר הקורס.
- כל סטודנט נדרש להגיש עבודות לבד. אין לעבוד בזוגות או בקבוצות גדולות יותר.
- **אין להעתיק עבודות או חלקי עבודות מתלמידים אחרים או מהאינטרנט. אסור השימוש בChatGPT לכתיבת קוד עבורכם.**
- אין לשלוח עבודות או קטעי עבודות לתלמידים אחרים. מותר ואף מומלץ לדון בבעיות מימוש וכן להציע פתרון לבעיות מקומיות. אך אין לשלוח, לשתף או להראות קטע קוד הפותר את הבעיה.
- שימו לב כי העבודה תיבדק אוטומטית, אך תיבדק גם צורת כתיבה של הקוד שלכם (כפי שיפורט בהמשך) באופן ידני ובצורה מדגמית. ועל כן, לאחר סיום תקופת ההגשה הציון עשוי לרדת לאחר הבדיקה הידנית המדגמית, במידה והעבודה לא כתובה על פי הסטנדרט שמפורט במסמך זה.
- שאלות לגבי העבודה יש לשאול בפורום באתר הקורס או בשעות קבלה של ענבל רושנסקי.
- תיאור המחלקות הנתונות והדרושות למימוש מופיע בסוף המסמך.
- יש לקרוא את כל המסמך טרם תחילת העבודה.

נושאי העבודה: מערכים , עץ בינארי והכרת ג'אווה.

מבוא:

במדעי המחשב, ביצוע שאילתות על נתונים זו אחת מהפעולות הבסיסיות שיש לבצע. מנסים לשפר תמיד את השליפה של השאילתות בעזרת שימוש במבני נתונים שונים. עץ סגמנטים, הוא מבנה נתונים המשתמש לאחסון מידע סיכומי על תתי קטעים בתוך הנתונים שלנו. בצורה זו ניתן לתשאל את העץ שאילתות בנוגע לטווחים על פני הנתונים שבו ולקבל תשובות באופן מהיר הרבה יותר מאשר לעבור בצורה איטרטיבית על כל הנתונים במקטע. עץ סגמנטים מספק גמישות כי הוא מאפשר שינויים מהירים של הנתונים ומאפשר למצוא מידע חשוב כמו סכום של רכיבי נתונים עוקבים, מינימום ומקסימום בטווחים שונים.
אחד המאפיינים החשובים של עץ סגמנטים הוא שהוא דורש כמות לינארית של זיכרון, וכל שאילתה על העץ מקבלת מענה בזמן ריצה של log(n), כאשר n זה מספר הנתונים אשר שמורים במבנה.

בעבודה זו תממשו מבנה נתונים מסוג עץ סגמנטים (segment tree) אשר מחזיר מקסימום, מינימום וסכום על פני טווח מסוים בנתונים ובנוסף מאפשר עדכון של ערך הנמצא במערך הנתונים שלנו לערך אחר בהתאם לרצון המשתמש.
את עץ הסגמנטים תממשו בשתי דרכים שונות. בייצוג בעזרת מערך ובייצוג בעזרת עץ.
בנוסף תממשו מבנה נתונים (number analyzer) אשר מבצע מניפולציות בעזרת עצי הסגמנטים על אוסף של מספרים מהמשתמש.

חלק ראשון: בניית עץ סגמנטים : Segment Tree

בחלק זה של העבודה תממשו מבנה נתונים בשם segment tree. על מנת לבצע זאת, תקבלו מערך המכיל מספרים שלמים (integers) ובעזרת מערך זה תחשבו את ערכי קודקודי העץ.

הסבר על עץ סגמנטים:

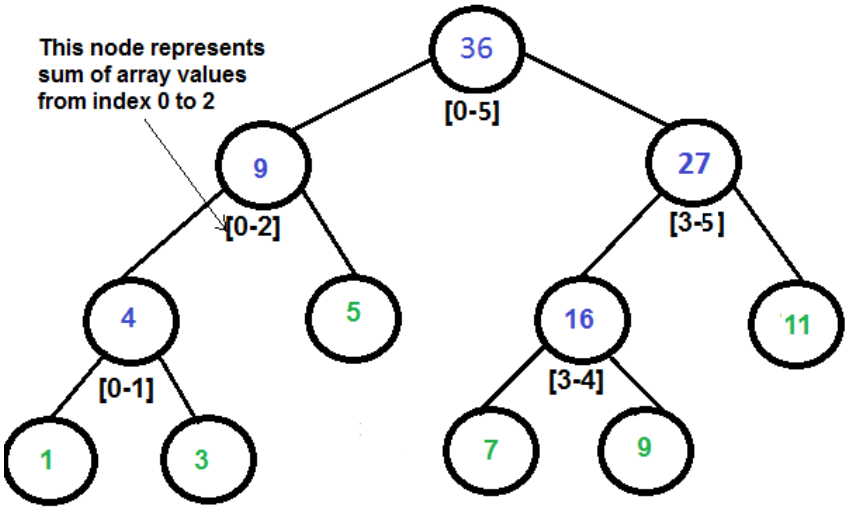
בניית עץ הסגמנטים כרוכה בחלוקה רקורסיבית של מערך הנתונים למערכים קטנים יותר, עד שכל מערך מכיל מספר בודד.
כל צומת עלה בעץ מייצגת אלמנט בודד של הנתונים מן המערך, וכל צומת שאינו עלה מייצג טווח של אלמנטים במערך. הערך של הצומת שאינו עלה מחושב על ידי שילוב ערכי הצמתי צאצאים שלו.
פעולת השילוב יכולה להיות שונה בהתאם לצורך. לדוגמא, מציאת מקסימום, מינימום, סכום וכו'.

לדוגמא, נניח כי קיבלנו את אוסף הערכים : [1,3,5,7,9,11] (חשוב לציין כי הערכים לא צריכים להיות ממויינים בשום צורה, זו סתם דוגמא). ניתן לראות כי האינדקס הראשון הוא 0, והאחרון בדוגמא זו הינו 5. אנו רוצים לבנות עץ סגמנטים כדי לתמוך בשאילתות טווח יעילות. לדוגמא, מה הסכום בין אינדקס 1 ל- 3 (תמיד כולל) והתשובה במקרה זה תהיה 15 (7+5+3).
למטרה זו נבנה את העץ. אנו נבנה את העץ באופן רקורסיבי מלמטה למעלה.

בכל קודקוד של העץ אנו בעצם מחלקים את טווח האינדקסים הנוכחי לשניים, ויוצרים צמתים לכל חצי. לדוגמא, קודקוד העץ יכיל את כל האינדקסים מ-0 ועד 5. ברמה הראשונה של העץ נחלק את הטווח [0:5] לשני חצאים: [0:2] ו [3:5]. אנו יוצרים שני צמתים בעץ כדי לייצג את הטווחים האלו ומגדירים את הערכים שלהם להיות במקרה של סכום, סכום של הטווח. כלומר, הצומת המתאים ל [0:2] יכיל את הסכום 9, והערך המתאים ל [3:5] יכיל את הסכום 27.

ממשיכים לבנות את העץ באופן רקורסיבי, וחוזרים על תהליך הזה עבור כל מחצית מכל טווח של איברים שנמצא בקודקוד עד שנבנה **עץ בינארי מלא (עץ בינארי מלא הוא עץ בו לכל צומת שאינו עלה יש שני בנים)** . בסוף בצורה זו, שורש העץ יכיל את סכום האיברים של המערך כולו.

עץ הסגמנטים המתאים לדוגמא (עבור סכום) יראה בצורה הבאה :



Segment Tree for input array {1, 3, 5, 7, 9, 11}

אנו בונים את העץ באופן רקורסיבי מהעלים לכיוון השורש, בצורה דומה לאופן שבו ראינו בתרגול הראשון בבניית עץ הטורניר.

יש שתי דרכים לייצג את העץ ולממש את הפונקציונליות של עץ הסגמנטים:

1) ייצוג בעזרת מבנה של עץ הכולל קודקודים ומצביעים לבנים
2) ייצוג בעזרת מבנה של מערך המדמה את הצורה של העץ

בעבודה זו, נממש את עץ הסגמנטים בשתי הצורות.

א) **מימוש העץ בעזרת ייצוג של מבנה של עץ:**

במימוש שלנו, לכל צומת בעץ קיימים השדות הבאים: (שימו לב כי הclasses של node בעץ נתונים לכם באופן מלא. מומלץ לעבור על הקבצים במקביל לקריאת העבודה)

Start- האינדקס הראשון בטווח שמיוצג על ידי קודקוד זה

end – האינדקס האחרון בטווח שמיוצג על ידי קודקוד זה

min – מייצג את הערך המינימלי בטווח [start:end]

max – מייצג את הערך המקסימלי בטווח [start:end]

sum – מייצג את סכום הערכים בטווח [start:end]

leftChild – הNode השמאלי המיידי של הקודקוד

rightChild – הNode הימני המיידי של הקודקוד.

הקודקודים יכילו את כל המידע אצלנו במימוש של העץ, אולם נבנה 3 מחלקות קונקרטיות לעצים השונים, כך שכל אחד יעשה שימוש רק במידע אחד מתוך הקודקוד (מינימום, מקסימום או סכום).

תיאור האלגוריתם הרקורסיבי, לייצור עץ סגמנטים בעזרת מבנה של עץ (קודקודים ובנים) בהינתן פרמטרים ראשוניים של מערך Arr בגודל N, אינדקס התחלה (0) ואינדקס אחרון:

- אם האינדקס ההתחלתי שווה לאינדקס האחרון (כלומר אנחנו בדיוק על ערך אחד וצריכים לייצג עלה בעץ) בנה קודקוד חדש שיקבל את אינדקס ההתחלה, הסוף, ויהיה עלה (כלומר ללא בנים) וכל הערכים הסיכומיים שלו (מינימום, מקסימום וסכום), שווים לערך שלו.
- אחרת: (אנחנו בקודקוד שאינו עלה)
- חשב את האינדקס האמצעי (ערך תחתון במקרה שהכמות לא מתחלקת ב-2)

-קרא לפונקציה זו בצורה רקורסיבית על החצי השמאלי (מהאינדקס התחלה ועד האמצע) ושמור את הערך החוזר כבן שמאלי של הקודקוד הנוכחי

- קרא לפונקציה זו בצורה רקורסיבית על החצי הימני (מהאינדקס אמצע+1 ועד לאינדקס האחרון) ושמור את הערך החוזר כבן ימני של הקודקוד הנוכחי.

-חשב ושמור בקודקוד הנוכחי את הערך המינימלי מבין הערכים המינימלים של הבן הימני והשמאלי שנוצרו.

-חשב ושמור בקודקוד הנוכחי את הערך המקסימלי מבין הערכים המקסימלים של הבן הימני והשמאלי שנוצרו.

-חשב ושמור בקודקוד הנוכחי את הסכום של הבן הימני והשמאלי שנוצרו.

-החזר את הקודקוד הנוכחי עם הערכים השמורים.

כעת, לאחר שבנינו את העץ, ונרצה לבצע שאילתה על טווח מסויים, אנו נשלוף את המידע המתאים לנו על פי הטווח הרצוי.

תיאור האלגוריתם הרקורסיבי, לשליפת מידע בעזרת מבנה של עץ הסגמנטים בהינתן פרמטרים ראשוניים של Node התחלתי (root), אינדקס התחלה ואינדקס סיום עבור הטווח:

-אם האינדקס התחלה וסיום של הקודקוד הנוכחי מוכל באופן מלא בתוך האינדקס התחלה וסיום שהתקבל בפונקציה, החזר את הקודקוד על כל ערכיו.

-אחרת:

-מצא את האינדקס האמצעי בטווח של הקודקוד הנוכחי. (ערך תחתון במקרה שהכמות לא מתחלקת ב-2)

אם אינדקס הסיום שהתקבל בפונקציה קטן או שווה לערך האמצעי של הטווח (סימן שצריך לקחת ערכים מתת העץ השמאלי המושרש בקודקוד הנוכחי של עץ הסגמנטים) : נקרא לפונקציה זו בצורה רקורסיבית על הבן השמאלי של הקודקוד הנוכחי, עם ערכי ההתחלה והסוף שהתקבלו בפונקציה.

אחרת, אם הערך התחלה שהתקבל בפונקציה, גדול יותר מהערך האמצעי של הטווח (סימן שצריך לקחת ערכים מתת העץ הימיני המושרש בקודקוד הנוכחי של עץ הסגמנטים) : נקרא לפונקציה זו בצורה רקורסיבית על הבן הימני של הקודקוד הנוכחי, עם ערכי ההתחלה והסוף שהתקבלו בפונקציה.

אחרת (מקרה זה אומר שהקודקוד הנוכחי מכיל חלק מהטווח המבוקש מימינו, וחלק משמאלו):

תקרא פעמיים בצורה רקורסיבית לפונקציה זו. פעם אחת עם הבן השמאלי, כאשר אינדקס ההתחלה נשאר כפי שהתקבל, ואינדקס הסיום מוחלף עם האינדקס האמצעי שנמצא בפונקציה. ופעם אחת עם הבן הימני, כאשר אינדקס ההתחלה מוחלף עם האינדקס האמצעי+1 והאינדקס הסיום נשאר כפי שהתקבל. נשמור את הערכים החוזרים כתוצאה שמאלית וימנית בהתאמה.

- נחשב את הערך המינימלי מהמינימום בין שתי התוצאות שהתקבלו.

-נחשב את הערך המקסימלי מהמקסימום בין שתי התוצאות שהתקבלו.

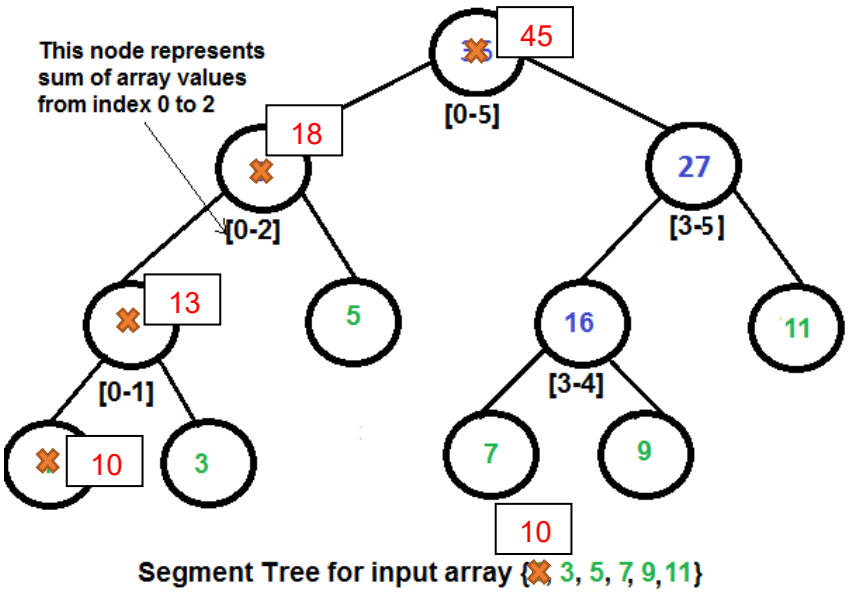
-נחשב את הסכום מהסכום של שתי התוצאות שהתקבלו.

-נחזיר node חדש מתחילת האינדקס הנוכחי של הקודקוד ועד סוף האינדקס הנוכחי של הקודקוד, בעל הערכים שחישבנו (מינימום, מקסימום וסכום).

אלגוריתם נוסף שיש לבצע עבור עץ זה הוא **עדכון ערך מסויים מתוך המערך**. פונקציה זו תקבל אינדקס של איבר במערך וערך (int) חדש, ותחליף את האיבר במערך המקורי (עלי העץ) באינדקס הנתון עם הערך החדש. חשבו איך לבצע את פעולה זו בצורה רקורסיבית.

בדומה למציאת שאילתת טווח, בהתאם לטווח של הקודקוד הנוכחי שאתם נמצאים בו יש לבצע set לערכים הסיכומים במידת הצורך.

לדוגמא, עבור הקריאה update עבור המערך ההתחלתי מהדוגמא הקודמת [1,3,5,7,9,11], האינדקס 0 ועם הערך החדש 10, המערך לאחר השינוי יהיה [10,3,5,7,9,11], והתשובה לשאילתת סכום עבור טווח [0:5] יהיה 45 לאחר העדכון. העץ לאחר העדכון יראה בצורה הבאה (שימו לב שהשינויים שיש לבצע הם רק במסלול שבו עובר הערך עם השינוי):



הסתכלו על הקובץ TESTER שניתן לכם לדוגמאות נוספות כיצד בניה של העץ, שאילתות טווח, עדכון והדפסת העץ צריכות להראות בכל אחד מהמימושים השונים.

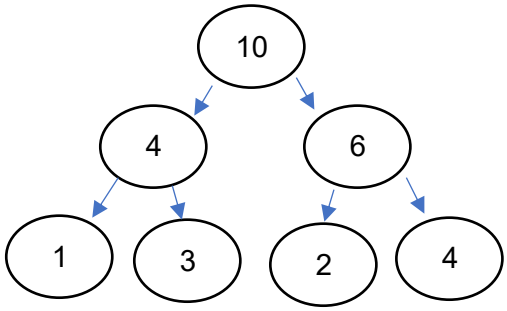
ב) מימוש העץ בעזרת מבנה של מערך :

עליכם לבנות את הפונקציונליות של עץ הסגמנטים פעם נוספת, הפעם כאשר המימוש עצמו יחזיק את מבנה הנתונים בתור ייצוג של מערך ולא בתור עץ פיזי עם קודקודים.
בחלק הזה, נדון באיך לייצג עץ סגמנטים בעזרת מערך ונדגים כיצד לנווט בין מבנה הנתונים (שהוא עץ) לייצוג של הקודקודים השונים במערך.

באופן כללי, ניתן לאחסן עץ ביעילות כמערך, כאשר כל תא במערך מייצג צומת, וניתן לגשת בקלות אל הילדים וההורה של כל קודקוד באמצעות האינדקסים בתוך המערך. כדי לייצג את עץ הסגמנטים כמערך, עבור קלט של מערך בגודל N, גודל מערך עץ הסגמנטים יהיה:
$\text{Size} = 2 * 2^{(\lceil \log_2 N \rceil)} - 1$. זאת על מנת להבטיח מספיק מקום לכל כמות הקודקודים שיש בהם צורך בעץ. כמות קודקודים אלו מייצגת עץ בינארי מושלם (הגדרה לעץ זה בהמשך העבודה).

כאשר מייצגים את עץ הסגמנטים כמערך, צומת השורש מאוחסן באינדקס 1 .(ראו הערה בהמשך לגבי מימוש בjava).
עבור על צומת אחר בעץ, נניח בעל אינדקס 'i' הבן השמאלי שלו מאוחסן באינדקס i*2, והבן הימני שלו מאוחסן באינדקס i*2+1. על מנת להגיע מצומת 'i' כלשהי לאבא המיידי שלו, נוכל למצוא את האב בצומת באינדקס $\left\lfloor \frac{i}{2} \right\rfloor$.

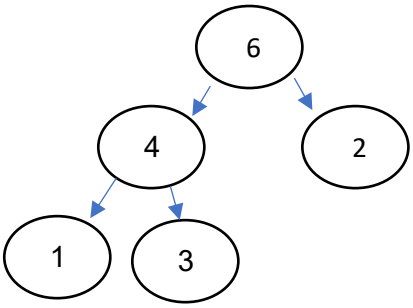
לדוגמא, נניח כי יש לנו את המערך הבא: [1,3,2,4] , כאשר במקרה זה N=4.
גודל המערך המחזיק את העץ יהיה בגודל $2 * 2^{(\lceil \log_2 4 \rceil)} - 1 = \mathbf{7}$.
העץ שנקבל עבור "סכום" יראה בצורה הבאה:



המערך המייצג את העץ יראה בצורה הבאה:

[10,4,6,1,3,2,4]

לעומת זאת, עבור המערך הבא: [1,3,2] , כאשר במקרה זה N=3,
גודל המערך המחזיק את העץ גם במקרה זה יהיה בגודל $2 * 2^{(\lceil \log_2 3 \rceil)} - 1 = \mathbf{7}$.
העץ שנקבל עבור "סכום" יראה בצורה הבאה:



המערך המייצג את העץ יראה בצורה הבאה:

[6,4,2,1,3, - , -]

בניגוד למימוש בעזרת מבנה של עץ, בו בכל קודקוד החזקנו את כל המידע על אותו סגמנט (מינימום מקסימום וסכום), פה נבצע את המימוש בעזרת מערך של integers, ולכן כל סוג של עץ יחזיק סוג אחד בלבד של מידע סיכומי.

<u>שימו לב:</u>

1) גודל המערך שנחזיק יהיה מתאים לעץ בינארי מושלם. עץ בינארי מושלם הוא עץ בינארי מלא (לכל צומת שאינו עלה יש שני בנים), בו כל העלים הם מאותה רמה.
אולם, עבור N שאינו חזקה של 2, לא יהיה צורך בכל המקומות במערך. נשים לי כי בשכבה האחרונה לא כל הקודקודים יהיו מלאים. על מנת לממש זאת בJAVA, יש לתת ערך דיפולטיבי (לדוגמא המספר הכי גדול\קטן שניתן להכיל בint) שיבחין בין קודקוד עם ערך תקין במערך לבין קודקוד עם ערך לא תקין במערך.

2) בג'אווה מערכים מתחילים מאינדקס 0, ואולם על מנת שיעבוד האינדקסים של המערך צריכים להתחיל מ- 1. יש לבצע התאמות על מנת לעבוד באופן טכני עם מערך זה.
בשאילתות טווח עבור עץ הסגמנטים, גם עבור מימוש העצים וגם עבור מימוש המערך, האינדקס הקטן ביותר (הראשון) שניתן לקבל בפונקציית טווח הינו 0.
הסתכלו על הקובץ TESTER שניתן לכם לדוגמאות נוספות כיצד בניה של העץ, שאילתות טווח, עדכון והדפסת העץ צריכות להראות בכל אחד מהמימושים השונים.

לפני שאתם ממשיכים את העבודה קראו היטב את הדוקומנטציה בקבצים שניתנו לכם, הבינו היטב את הדוגמאות, וכיצד עוברים בין הקודקודים בעזרת האינדקסים של המערך, ובעזרת המימוש של העץ.

חלק שני: בניית Number Analyzer

בחלק השני של מטלה זו, תבנו מחלקה של NUMBER ANALYZER, שתמנף את עצי הסגמנטים שבניתם קודם לכן. המטרה העיקרית של מחלקה זו היא לבצע ביעילות פעולות שונות על מערך מספרים ולספק למשתמשים ממשק אחיד לגישה לפונקציות שונות.
מחלקת NUMBER ANALYZER תממש את ממשק Iterator וגם את ממשק Comparator. על ידי הטמעת ממשק Iterator, המשתמש יוכל לעבור על רכיבי המערך ברצף, ללא גישה ישירה למערך עצמו בו שמורים האיברים. בעוד שממשק Comparator יקל על השוואה מותאמת אישית עבור פעולות מיון או חיפוש. ההשוואה שנרצה לתמוך בה היא לפי רמת זוגיות של המספר. על פי ההשוואה של Comparator, מספר זוגי ייחשב כמספר שגדול יותר ממספר אי זוגי.עבור שני מספרים באותה רמת זוגיות, ההשוואה תעשה לפי הערך של המספרים. כלומר, במיון על פי ההשוואה של הcomperator, יופיעו קודם כל כל המספרים האי זוגיים ממויינים מהקטן לגדול, ולאחר מכן כל המספרים הזוגיים ממויינים מהקטן לגדול. שימו לב, בפונקציית ההשוואה, יוחזר 1 אם המספר השני קטן מהראשון, יוחזר 1- אם המספר הראשון קטן מהשני, ויוחזר 0 במידה והם שווים.
בנוסף מחלקה זו תתמוך גם במספר פעולות חיוניות, כגון getMax, getMin, getSum וupdate.
שלושת פונקציות ה-get , יקבלו שני אינדקסים מהמשתמש ויחזירו את הערך המקסימלי, מינימלי והסכום במערך בין שני אינדקסים אלו, בהתאמה. פעולת העדכון תקבל אינדקס במערך וערך חדש, ותאפשר למשתמש לשנות אלמנט אחד במערך ולעדכן אוטומטית את המידע המתאים במחלקה.

נממש את המחלקה הזו בשתי צורות, שתתאים לכל אחת מצורות מימוש העצים. פעם אחת נממש את המחלקהבעזרת עצי הסגמנטים הממומשים בעזרת מבנה של עץ, ופעם השניה נממש את המחלקה בעזרת עצי הסגמנטים הממומשים בעזרת מערך.

חלק ג' בדיקת הקוד

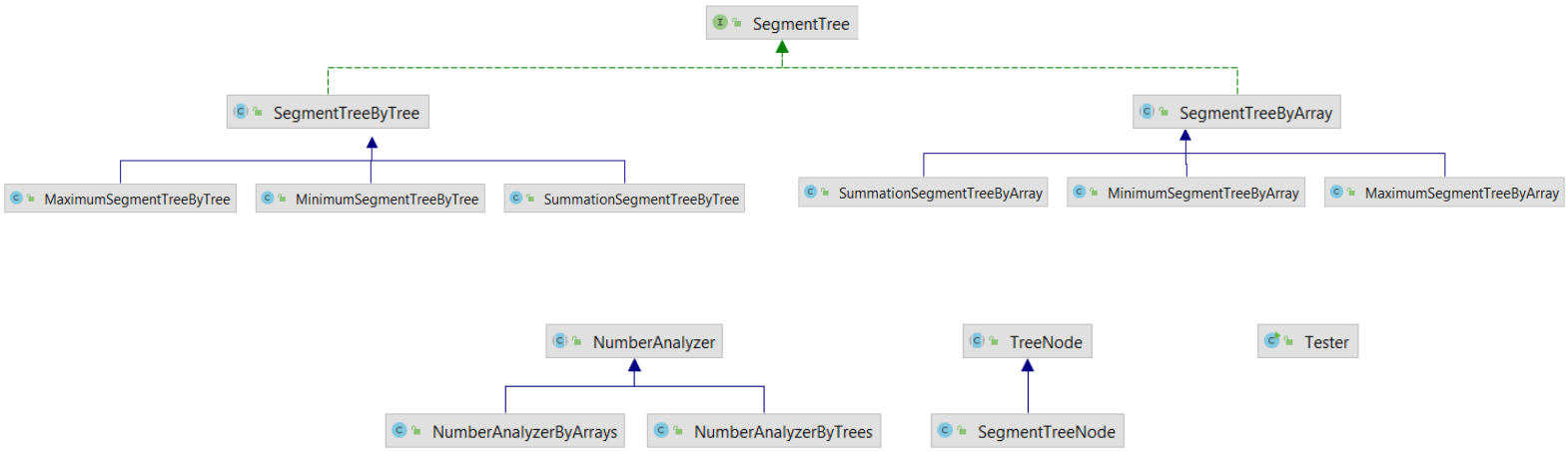
כחלק מהתרגיל, אתם תידרשו לכתוב בדיקות (tests) עבור הקוד שלכם, מעבר לבדיקות שיינתנו לכם במערכת ההגשה. פירוט של חלק זה נמצא תחת המחלקה Tester בחלק המפרט את המחלקות בתרגיל.

חשוב:

- יש להקפיד על קוד ברור וקריא, עם שמות משתנים משמעותיים. יש לכתוב דוקומנטציה לפני כל בלוק קוד לא טריויאלי ולפני כל פונקציה (יש להשתמש בפורמט המוכר של JAVA שראיתם בתרגול בשם JAVADOC), שמסבירה מה הבלוק/פונקציה עושה.
- **אין לייבא ספריות חיצוניות ובכללי אסור להשתמש בפקודה import באף אחד מהמחלקות שאתם מממשים (למעט שימוש ב import java.util.Comparator; import java.util.Iterator;** במחלקה של numberAnalyzer). שימו לב ש-eclipse לא מוסיף לכם בטעות import מיותר ותדאגו למחוק את כל ה-imports לפני הגשת העבודה, כיוון שיש כלי אוטומטי שבודק זאת, וירד על כך נקודות בציון הסופי.
- יש לכתוב את הקוד בצורה מיטבית, מה שכולל הוצאת קוד חוזר לפונקציות ואי שכפול של קוד מיותר, מחיקת קוד לא רלוונטי ולא רק הכנסתו להערה וכו'.
- במידה והקוד לא יעמוד בסטנדרט שנכתב לעיל, ויתפס בבדיקה המדגמית של הקוד, יורדו על כך נקודות בצורה משמעותית.
- אין צורך באף שלב של העבודה לבדוק את תקינות האינדקסים שמתקבלים בפונקציות. לא ינתן כחלק מהטסטים אינדקסים מחוץ לגבולות המערך (תמיד בין 0 ל- כמות האיברים פחות1) או שהאינדקס הגדול ינתן כקטן וכו.

פירוט המחלקות בתרגיל:

התרשים הבא הוא תרשים המחלקות שיש בעבודה (כולל את הניתנים לכם, את אלו שיש להרחיב ואת אלו שתצטרכו ליצור בעצמכם):



המחלקות שאנו מספקים לכם- אין לשנות מחלקות אלה – אתם לא מגישים אותם, ולכן שינוי בהם יגרור כישלון בקמפול הקוד בשרת הבדיקות:

`Interface SegmentTree`

(in the SegmentTree.java file)

A segment tree data structure that supports efficient range queries and updates for integers. The interface declares the basic functionality of a SegmentTree. The class itself contains all the functions with full documentation.

`abstract class TreeNode`

This class represents a node in a binary tree.

`class SegmentTreeNode`

This class represents a node in a segment tree that is implemented with a tree structure. This node contains summary information regarding the segment it represents including its indexes. This class extends TreeNode, and should be used to build the segment trees.

המחלקות שאתם צריכים לממש או שניתנו באופן חלקי – שימו לב שאין למחוק פונקציות אשר החתימה שלהם מופיעה בקובץ נתון. אין לשנות את החתימה גם כן – הדבר יגרור כישלון בקימפול הטסטים שלכם:

`abstract class SegmentTreeByTree`

An abstract class for a segment tree data structure implemented using a tree structure. Subclasses must implement the {@code queryRange} method to provide specific range query functionality.

`class MaximumSegmentTreeByTree ,`

`class MinimumSegmentTreeByTree`

`and class SummationSegmentTreeByTree`

These are the classes that extend the abstract class SegmentTreeByTree, and they implement the functions required for the return of maximum, minimum and sum respectively.
These classes should implement the toString function. To ensure proper execution of the toString function, it is important to perform a **pre-order traversal** of the tree. **Pre-order traversal** means that we visit the root node first, followed by the left subtree and then the right subtree. By doing so, we can take the members of the tree in the correct order. The elements should be surrounded by "[ ]" and exactly one space between each number and between the brackets.

`abstract class SegmentTreeByArray`

This abstract class represents a segment tree implementation using an array.
Subclasses must implement the {@code queryRange} method to provide specific range query functionality. This class should implement the toString function for its subclasses. Compared to the implementation using trees, in the implementation using an array, the toString will take the members according to their order in the represented array. You should go through the array from the first cell to the last. The elements should be surrounded by "[ ]" and exactly one space between each number and between the brackets. Do not add default values, instead of these values add "-" if there are no nodes in these places in the tree.

`class MaximumSegmentTreeByArray ,`

`class MinimumSegmentTreeByArray`

`and class SummationSegmentTreeByArray`

These are the classes that extend the abstract class `SegmentTreeByArray`, and they should implement the functions required for the return of maximum, minimum and sum respectively.

`abstract class NumberAnalyzer`

The NumberAnalyzer class provides an abstract implementation for analyzing a collection of integers.
It implements the Iterable interface to allow for iteration over the collection, and the Comparator interface to provide a comparison method for the values.

`class NumberAnalyzerByArrays , class NumberAnalyzerByTrees`

These are the classes that extend the abstract class `NumberAnalyzer`, and they implement the functions required to work with Arrays trees and Trees trees, respectively.

`class Tester`

The class `Tester` will help you test your code. You should add tests for every public method. Your tests should include extreme ("boundary") cases, as well as the obvious "middle of the road" cases. Remember to choose **diverse** test cases.
We suggest the following approach:

- Test basic methods before complicated ones.
- Test simple objects before objects that include or contain the simple objects.
- Do not test too many things in a single test case.

We also provide the file Tester.java. This file includes a main function that will be in charge of running all the tests for your code. A helper function that you will use is the following:

```
private static void test (boolean exp, String msg)
```

This function receives a Boolean expression and an error message. If the Boolean expression is evaluated as false, the function will print the error message to the screen. The function will also "count" for you the number of successful tests that you executed.

Please read carefully the code in this file. As you can see, we already provided a few tests to your code. You are required to add your own tests to check Part A, B and C of the assignment, so the output of running the `Tester`, will be:

```
All <i> tests passed!
```

Where `i` (the total number of tests) should be at least **50**! If you will have less than 50, your grade will be reduced.

<u>הנחיות הגשה:</u>

עליכם להגיש את כל המחלקות שצוינו לעיל (מלבד המחלקות שניתנו לכם באופן מלא) למערכת ההגשה, לפי הוראות ההגשה במודל. **יש להגיש את הקבצים הבאים:**

MaximumSegmentTreeByArray.java, MaximumSegmentTreeByTree.java,
MinimumSegmentTreeByArray.java, MinimumSegmentTreeByTree.java,
NumberAnalyzer.java, NumberAnalyzerByArrays.java, NumberAnalyzerByTrees.java,
SegmentTreeByArray.java, SegmentTreeByTree.java,
SummationSegmentTreeByArray.java, SummationSegmentTreeByTree.java,
Tester.java

הנחיות נוספות בנוגע להגשת עבודות יצורפו במסמך נפרד במודל.

שימו לב כי העבודות עוברות בדיקות מחמירות למציאת העתקות. הכלי מוטמע במודל והבדיקה תעשה בסיום תקופת ההגשה.

סטודנטים שיתפסו בהעתקה אחד מהשני, מהאינטרנט, מהדרייב וכו' יענשו בחומרה.

בהצלחה!!