<u>מבני נתונים עבודה 4</u>

מגישים: יותם דיקשטיין, אמיר ארבל.

טענות כלליות החשובות להמשך, על עצים חד-מימדיים כנ"ל:

1. **העץ הינו עץ מלא** (אין צמתים עם בן אחד) - נובע מאלגוריתם הבניה שמקנה לכל צמת שאינו עלה שני בנים. במימוש האלגוריתם שמרנו על תכונה זו בהוספה/הסרת עלים מהעץ. בתרגול הוכחנו כי בעץ מלא מספר העלים n, הוא כחצי ממספר הצמתים בעץ ולכן כמות הצמתים הכוללת בעץ היא 2n-1 (שזה $O(n)$).
2. כל הנקודות בעלי בעץ, מופיעות פעם אחת בדיוק בצמתים הפנימיים שאינם עלים (פרט למקסימום שלא מופיע כלל):
   - a. **המקסימום לא מופיע כלל:** המקסימום לא נמצא בתת עץ שמאלי של אף עלה אחר.
   - b. **נקודה לא תופיע יותר מפעם אחת:** נניח בשלילה שהנקודה x מופיעה יותר מפעם אחת בצומת פנימית. נבחר את הצומת התחתונה היותר שבה היא מופיעה - y. מהחוקיות בעץ, העלה של הצומת התחתונה הנ"ל הינו העלה הגדול ביותר בתת העץ השמאלי של y. תת עץ מלא ולכן, בפרט, יש לו בן ימני שגדול מ-x, נסמנו w. הנחנו בשלילה שיש צומת נוסף z, שמילה את הנקודה x. לכן z הינו אב קדמון של x, וש-x הוא בתת העץ השמאלי. מכאן שגם z גם אב קדמון ל-y ושגם y בתת העץ השמאלי של z (מתכונות עץ החיפוש). אם y בתת העץ השמאלי, אז גם w שגדול מ-x, בתת העץ השמאלי של z. לכן z צריך להכיל את w או נקודה שגדולה מ-w, **ובפרט לא את x.**
   - c. **כל נקודה מופיע פעם אחת לפחות (פרט למקסימום):** אף נקודה לא מופיעה יותר מפעם אחת. המקסימום לא מופיע כלל. ולכן, בכדי לקיים את התכונה שיש עוד n-1 צמתים פנימיים - כל נקודה (פרט למקסימום) על כל נקודה להופיע בדיוק פעם אחת.
3. **יהא n מספר העלים בעץ. גובה העץ h בבנייה, הינו** $O(log(n))$ **לכל היותר:** כלומר קיים c חיובי כך ש-$h \le c * log(n)$. **הוכחה** באינדוקציה על n מספר העלים:
   - i. **בסיס:** n=0 העץ הריק מקיים את התנאי לכל c חיובי, ובפרט לכל c>1 (למשל c=2). כנ"ל גם עבור n=1.
   - **ii. צעד האינדוקציה:** נניח כי הטענה נכונה לכל k שקטן מ-n העלים, כלומר לכל עץ בן פחות מ-2n-1 צמתים (n עלים, ו-n-1 צמתים פנימיים נלוים). בצעד הראשון באלגוריתם הבניה מסירים את האיבר האמצעי, מה שמשאיר שני עצים עם שיהיו עם $\frac{n-1}{2}$עלים (בעץ זוגי נקבל עץ אחד עם ערך עליון, והשני עם ערך תחתון). לאחר מכן, לעצים שייבנו יהיו לכל היותר $\frac{n}{2}$ עלים ערך תחתון. מהנחת האינדוקציה, גובה המקסימלי מביניהם הינו $c * log(\frac{n}{2})$, ולכן גובה העץ הינו $1 + c * log(\frac{n}{2})$. במקרה הבסיס לקחנו c=2.
     $$h = c * logn(\tfrac{n}{2}) \ + 1 \ = \ c * log(\tfrac{n}{2}) + log(2) \ \le c(log(\tfrac{n}{2}) \ + \ log(2)) \ = \ c * log(n)$$
     ולכן הטענה נכונה.

   נשים לב שמכיוון שאנו מניחים שלא קוראים להוספה/הסרה של עלים מהעץ יותר מ-logn פעמים, גובה העץ נשאר פרופורציונלי ל-logn מספר העלים (או הצמתים), גם אם יש לקחת קבוע גדול יותר. לכן מעתה, דברים שפרופורציונלים אסימפטוטית לגובה העץ, גם פרופורציונלים ל-log מספר העלים/צמתים.

מימוש העץ הדו מימדי:

כדי לממש עץ דו-מימדי, ראשית בנינו מחלקה של עץ חד מימדי שמממשת הרבה מהשיטות שבמשימה, ולאחר מכן עשינו אדפטציה לעץ דו מימדי, של מה שנדרש.
הערה: בעץ החד מימדי, תמכנו בחיפושים/הסרות/הוספות בהוספת אותו הנקודה, על אף שאין צורך כזה, על מנת שבעץ הדו-מימדי יהיה ניתן להוסיף נקודות שונות עם אותו פרמטר X או Y. אם נקודה מסוימת שווה לנקודה אחרת בפרמטר כלשהו, היא יכולה להופיע הן בעץ השמאלי, או הימני של אותה נקודה, בלי שהדבר יפגע בנכונות האלגוריתמים (ובפרט, תמיד בדקנו בחיפוש הטווח גם שיוויון).

מחלקת העץ החד-מימדי:

- שדות:
  - root - שורש העץ. יכיל טיפוס של OneDNode:
    - ה-OneDNode מכיל שדות data (עבור הנקודה), left, right, parent (עבור הקשרים לאב/בנים), ושדה leavesUnderNode (שמכיל את מספר העלים שנמצאים מתחת ל-node, כולל ה-node עצמו אם הוא עלה).
  - numOfLeaves - מספר העלים סה"כ בעץ.
  - Comp - comperator שמשמש להשוואה בין העלים.
- שיטות:
  - **בנאי** - בונה את העץ בדומה לאלגוריתם שניתן בהוראות. בנוסף, נותן לכל צומת את מספר העלים שתחתיו. סה"כ $O(nlog(n))$ זמן (מיון המערך בזמן זה. יתר הפעולות נקראות כמספר הצמתים בעץ שהוא $O(n)$).
  - **getAllPoints** - מחזיר את כל הנקודות מתחת לעץ. יש גרסה גם עבור כל node בנפרד. האלגוריתם יורד מה-node לכל הצמתים בעץ ומוסיף אותם למערך (בעזרת משתנה גלובלי). מכיוון שהפונקציה נקראת פעם אחת לכל צומת, וסך העבודה לכל צומת היא $O(1)$, <u>הפונקציה היא ב-$O(LeavesUnderNode)$ של הצומת (שהוא n עבור כל העץ).</u> נשים לב שמשיקולי פשטות הוספנו משתנה גלובלי שיכיל את המיקום הראשון במערך בו ניתן לשים את העלה. יכולנו, בלי להאריך אסימפטוטית את זמן הריצה - לעקוף בעיה זו גם ע"י העברת מחסנית.
  - **addPoint** - מוסיפה נקודה חדשה x לעץ. מוצאת ב-$logn$ את העלה y, הגדול ביותר שקטן או שווה לנקודה (בעזרת הקומפרטור), ומעדכנת תוך כדי את מספר העלים תחת הצמתים הנ"ל (בעזרת פונקצית עזר). אם הנקודה גדולה מכל העלים, אז פונקצית העזר תחזיר את המקסימום בעץ. חלק זה לוקח כגובה העץ זמן $O(log(n))$ (חיפוש בינארי). לאחר מכן, יוצרת ל-node הנ"ל שני בנים, וכשאחד מהם הוא העתק של y, ואחד מהם הוא x. לאחר מכן, עולה ומעדכנת את העלה הראשון שנקודה x היא המקסימלית בתת העץ השמאלי שלה (המקום הראשון שתת העץ של x הינו תת עץ שמאלי). <u>סה"כ $O(log(n))$ זמן</u>.
    הערה: שיטה זו מוסיפה לגובה העץ 1 לכל היותר. לכן אם מגבילים את עצמנו ל-logn קריאות, הפרופורציה בין גובה העץ למספר העלים/צמתים.
  - **remove Point -** מסירה את נקודה x מהעץ. ראשית - מחפשת את x בעץ ב-$O(log(n))$. אם הוא לא נמצא אז מחזירה false ומסיימת (<u>בכישלון $O(log(n))$ זמן</u>). אם לא, אז מחליפה את האב של x, בבן השני של אב זה (ובכך מעשית מוחקת את x ואת האב של x). לאחר מכן עולה בעץ, מעדכנת את השדה leavesUnderNode בכל צומת, ואם הצומת מכילה את x, אז מוצאת את המקסימום החדש בתת-העץ השמאלי שלה. <u>סה"כ $O(log(n))$ זמן בהצלחה</u>.
  - **existsPoint** - מחפש חיפוש בינארי נקודה לפי פרמטר ה-y שלה. ב-$O(log(n))$ זמן.
  - **numOfPointsInRange** - מחזירה מספר הנקודות בטווח [min,max] כולל, באופן הבא:

- מחזיקים משתנה ans=0, מצביע שמחפש את min, ומצביע שמחפש את max בחיפוש בינארי: כל עוד שני המצביעים נמצאים באותו מקום בעץ, ממשיכים לרדת.
- לאחר הפיצול ממשיכים עם כל מצביע בנפרד באופן סימטרי (נתאר עבור המצביע המינימלי):
  - המצביע המינימלי ממשיך שמאלה כל עוד הוא גדול/שווה מהמינימום, ומוסיף ל-ans את leavesUnderNode, של תת העץ הימני של ה-nodeים אותם הוא עובר.
  - כשהוא מגיע ל-node שקטן מהמינימום, הוא מתקדם בתת העץ הימני שלו כל עוד הערך min גדול מהערך ב-node.

שני המצביעים נעצרים בסוף העץ, לכן האלגוריתם לוקח $O(h) = O(log(n))$ זמן.

- **getPointsInNode** -מחזיר את הנקודות מתחת ל-node מסוים בטווח מסוים (הועמסה גרסה כללית לכל העץ ללא ציון node). הפונקציה פועלת בדומה ל-**numOfPointsInRange**, פרט לשינויים הבאים:
  - במקום int, סופרים את כמות הנקודות בטווח, ומאתחלים מערך בשם ans בגודל מספר הנקודות ($O(log(n) + k)$ זמן, כש-k מספר הנקודות בטווח).
  - במקום הוספת leavesUnderNode של ה-nodeים באלגוריתם לעיל, קוראים ל-getAllPoints בכל node רלוונטי ומעתיקים למערך ans את העלים שחזרו מ-getAllPoints (סה"כ $O(k)$ לכל הקריאות יחד לפונקציה זו).

  סה"כ הזמן שמתקבל הוא $O(log(n) + k)$.
- **numOfPointsInHalfPlaneY** - מחזיר את מספר הנקודות בין Y נתון למינימום/מקסימום בעץ. מוצא את המינימום/מקסימום ב-$O(log(n))$ זמן (חיפוש בינארי), וקורא ל-numOfPointsInRange בטווח של Y והמינימום/מקסימום. סה"כ $O(log(n))$ זמן.

מחלקת העץ הדו-מימדי:

העץ הזה נבנה בדומה לעץ חד-מימדי כשהמיון הוא לפי מפתח x, אבל בכל node ישנו מצביע לעץ חד-מימדי מהמחלקה הנ"ל שמכיל את כל הנקודות, ממויינות לפי פרמטר y.

- שדות:
  - root - שורש העץ, מכיל node מטיפוס TwoDNode שמכיל פרט לערכים של OneDNode, מצביע לעץ חד-מימדי yTree.
  - Comperator x,y - מכילים קומפרטורים לשני הפרמטרים.
- שיטות:
  - **בנאי** - בונה את העץ בדומה לאלגוריתם בהוראות. העתקת המערך ומיונו לפי x ו-y לוקחת $O(nlogn)$ זמן. פונקצית ה-partition לוקחת $O(n)$ זמן (יש בה כ-n השוואות), ובניית העץ החד מימדי (ללא מיון, שכן, ה-partition יציב), לוקחת גם היא $O(n)$ זמן. לכן בניית העץ הדו מימדי (לאחר מיון המערכים) הינה במקרה הגרוע:

    $$T(n) = 2T(\frac{n}{2}) + O(n)$$

    בכיתה הוכחנו שהפתרון למשוואה זו היא $O(n)$, לכן סה"כ הזמן שלוקח לבנות את העץ הוא $O(nlogn)$.
  - **ExistsPoint** - מחפש את הנקודה העליונה בעץ בה ערך ה-x הינו של הנקודה בקלט, בחיפוש בינארי ($O(logn)$), ולאחר מכן מחפש בתת העץ שלו אם הנקודה קיימת (לפי פרמטר ה-y) בחיפוש בינארי נוסף. סה"כ $O(log(n))$ זמן.

- **addpoint** - פועלת בדומה לעץ החד מימדי, אבל במהלך חיפוש הנקודה, בכל node שאליו המצביע שמחפש את הנקודה מגיע, קוראים להוספת הנקודה בעץ החד-מימדי של ה-node. סה"כ $O(log(n))$ קריאות לפונקציה של $O(log(n))$ זמן (ויתר הפעולות גם ב-$O(log(n))$ או פחות), <u>כלומר</u> $O(log^2(n))$.
- **removePoint** - מחפש את הנקודה בעזרת existsPoint (ב-$O(log(n))$ זמן). אם הנקודה לא נמצאת (בכישלון) מחזיר false. אם הנקודה נמצאת, פועל בדומה לפונקציה בעץ החד-מימדי, אבל בירידה לנקודה שמסירים, בכל node אליו המצביע מגיע, קוראים ל-removePoint בעץ החד-מימדי של ה-node. ובדומה לעיל, <u>זמן הריצה הינו</u> $O(log^2(n))$.
- **numOfPointsInRectangle** - פועלת בדומה לעץ החד-מימדי, עם פרמטר ה-X במקום פרמטר ה-Y, אולם במקום פעולת הוספת ה-leavesUnderNode למשתנה ans, היא קוראת לפונקציה **numOfPointsInRange**, לעץ החד-מימדי של ה-node ומוסיפה את התשובה. לכן, בדומה לאלגורתמים לעיל, <u>זמן הריצה הינו</u> $O(log^2(n))$.
- **getPointsInRectangle** - פועלת בדומה לעץ החד מימדי:
  - חיפוש מספר הנקודות ואתחול המערך בגודל זה לוקח $O(k + log^2(n))$
  - בחיפוש הנקודות, במקום לקרוא ל-getAllPoints, קוראים ל-**getPointsInRange** של העץ החד מימדי בכל node. זמן הריצה הינו לכל קריאה היא $O(t + log(h_x))$ (כש-t הוא מספר הנקודות בטווח תחת כל עץ ספציפי, x הוא ה-node שבו קראו לפונקציה, ולכל היותר ישנן $O(log(n))$ קריאות). לכן זמן הריצה הינו:

$$T(n) = O(k + log^2(n)) + \sum_{t \in Tree} O(t + log(n)) = O(k + log^2(n)) + O(k) + O(klogn)$$

$$T(n) = O(k + log^2(n)) + (O(\sum t + \sum log(h_x))$$

נשים לב שסכום כל ה-t הוא k, וש- $\sum log(h_x) \le \sum log(n)$, ומכיוון שיש לכל היותר $O(log(n))$

קריאות אזי $\sum log(n) \le log^2(n)$. <u>מכאן שזמן הריצה בסה"כ הוא</u> $O(k + log^2(n))$ <u>בכל מקרה.</u>

- **numOfPointsInHalfPlaneX** - פועלת בדיוק כמו בעץ החד-מימדי.
- **getAllPoints** - פועלת בדיוק כמו בעץ החד-מימדי.