עבודה 2 במבני נתונים – בעיית המטוסים

מגישים: ניב לוי ודין אברג'ל

על מנת לממש את הממשק DT, השתמשנו בשתי רשימות מקושרות דו-כיווניות המכילות קונטיינרים, אחת אשר מכילה את כל הקונטיינרים שהוכנסו למבנה הנתונים כך שהנקודות שלהם ממוינות לפי ציר ה-X והשנייה לפי ציר ה-Y. כמו כן, בין הרשימות ישנם מצביעים כלומר מקונטיינר המכיל את הנק' (x1,y1) הנמצאת בציר ה-X במקום ה-i קיים מצביע לאותה נקודה ברשימה השנייה.

השיטה `addPoint` :

השיטה מקבלת נקודה ויוצרת לה שני קונטיינרים **O(1),** יוצרת מצביע בין הקונטיינרים **O(1)** ואחר כך מכניסים אותם לרשימות, אחד לרשימה הממוינת לפי ציר ה-X ואחת לפי ציר ה-Y. פעולת ההכנסה מתבצעת ע"י הפונקציה insert(container, comparator X/Y) שבמקרה הגרוע תכניס את הקונטיינר לסוף הרשימה (במידה וערך ה-X/Y הוא הכי גדול) כלומר תעבור על כל הרשימה – **O(n).**

זמן ריצה: O(n)

הפונקציה `getPointsInRangeRegAxis`:

השיטה רצה על אחת הרשימות בהתאם לערך ה-axis שהוכנס, פעם ראשונה היא רצה כדי למנות את כמות האיברים שנכנסים בין ערך המינימום למקסימום, במקרה הגרוע **O(n)**. לאחר מכן יוצרת מערך בגדול המתאים, ולבסוף עוברת על הרשימה פעם נוספת ומכניסה את האיברים שנמצאו בטווח לתוך המערך המוחזר **O(n)**.

זמן ריצה: O(n)

השיטה `getPointsInRangeOppAxis` :

השיטה פועלת בדיוק כמו הפונקציה `getPointsInRangeRegAxis` ונבדלת בכך שבהינתן ציר axis הפונקציה רצה על ציר axis!, פעם אחת בכדי למנות את כמות האיברים בטווח **O(n)** ופעם שנייה מכניסה את האיברים המתאימים למערך המוחזר **O(n)**.

זמן ריצה: O(n)

השיטה `getDensity`:

השיטה מודדת את ערכי X ו-Y המינימליים והמקסימליים, מקבלת אותם באמצעות המצביעים של הרשימות: start – לאיבר הראשון, שהוא המינימלי, ו-end – לאיבר האחרון שהוא המקסימלי. **O(1).** לאחר מכן מבצעת את חישוב הצפיפות לפי הנוסחה הנתונה - **O(1).**

זמן ריצה: O(1)

השיטה `narrowRange` :

השיטה יוצרת שני מצביעים, אחד לסוף הרשימה בציר X ואחד לתחילת הרשימה בציר X. השיטה עוברת על כל הקונטיינרים עד אשר מגיע המצביע העליון לערך המקסימום שהוכנס והמצביע התחתון מגיע לערך המינימום שהוכנס. המחיקה מציר ה-X מתבצעת ע"י שיטת `delete` של `LinkedList` המבוצעת בזמן **`O(1)`** והמחיקה מציר ה-Y מתבצעת ע"י קבלת הקונטיינר בציר ה-Y ע"י `getOpp` בזמן **`O(1)`** ולאחר מכן מוחקת גם את הקונטיינר הזה ע"י `delete` של `LinkedList`. בסך הכל פעולת המחקה לכל קונטיינר מתבצעת בזמן **`O(1)`** והמצביעים רצים על הרשימה בזמן **`O(n)`** (במקרה הגרוע ייתכן שהמצביעים ירוצו על כל הרשימה).

זמן ריצה: `O(n)`

השיטה `getLargestAxis` :

כמו בשיטה `getDensity` נקבל את ערכי המינימום והמקסימום של X ו-Y באמצעות המצביעים של הרשימות **O(1)** . לאחר מכן מחזירה את האם הפרש בין X מקסימלי למינימלי > הפרש בין Y מקסימלי למינימלי, חישוב של **O(1)**.

זמן ריצה: O(1)

<u>השיטה Split(int value, Boolean axis):</u>

השיטה מקבלת משתנה value וציר נתון ומחזירה מערך בגודל 2, התא הראשון במערך מכיל את אוסף הקונטיינרים שקטנים מ-value כך שהקונטיינר הראשון הוא האיבר הכי קרוב לערך value, ניתן להתקדם ע"י prev ולרדת עד הערך המינימלי שקיים במבנה הנתונים (בהתאם לציר שניתן לשיטה ספליט). התא השני במערך מכיל את אוסף הקונטיינרים שגדולים מ-value כך שהקונטיינר הראשון הוא האיבר הכי קרוב לערך value, ניתן להתקדם ע"י next ולעלות עד הערך המקסימלי שקיים במבנה הנתונים.

כמו כן, הפונקציה בודקת 2 מקרי קיצון לגבי הערך value: אם הוא גדול מהערך המקסימלי במבנה הנתונים אז נחזיר את כל הרשימה בתא הראשון ו-Null בתא השני. לחילופין, אם הוא קטן מהערך המינימלי במבנה הנתונים נחזיר בתא הראשון Null ובתא השני נחזיר את כל הרשימה.

<u>ניתוח זמן ריצה של ספליט:</u>

השיטה רצה באמצעות שני מצביעים: אחד שיורד מסוף הרשימה ואחד שעולה מתחילת הרשימה, שני המצביעים מתקדמים ביחד עד אשר אחד מהם חוצה את הערך value (או מלמעלה ע"י tailPointer או מלמטה ע"י headPointer), שני המצביעים מתקדמים ביחד לכן בסך הכל עברנו על 2|C| איברים מתוך הרשימה. בנוסף מתבצעות פעולות אטומיות כמו יצירת מערך, יצירת המצביעים והשמות בתאי המערך של 2 קונטיינרים בלבד, כמו כן, מתבצעת בדיקה של מקרי הקיצון בהן value הוא ערך גדול או קטן מערכי המקסימום או המינימום שבמבני הנתונים (בהתאמה). כל הפעולות הללו מתבצעות בזמן של **O(1)**. ובסה"כ השיטה רצה בזמן של **O(2|C|)**, אנחנו מתעלמים מהקבוע 2 ולכן השיטה רצה בזמן **O(|C|)** כנדרש.

פסודו קוד:

<u>Split (int value, Boolean axis)</u>

1. Create a pointer for the head of the given axis list. (headPointer)
2. Create a second pointer for the tail of the given axis list. (tailPointer)
3. Create an array of size 2
4. If (headPointer coordinate > value)
    a. array[0] = null
    b. array[1] = headPointer
5. if (tailPointer coordiante < value)
    a. array[0] = headPointer
    b. array[1] = null
6. While (headPointer coordinate < value AND tailPointer coordinate > value) do
    a. headPointer = headPointer.next
    b. tailPointer = tailPointer.prev
7. if (headPointer coordinate >= value)
    a. array[0] = headPointer.prev
    b. array[1] = headPointer
8. else array[0] = tailPointer
9. array[1] = tailPointer.next
10. return array

השיטה nearestPairInStrip(Container container, double width, Boolean axis):

השיטה מעתיקה את שתי הרשימות (שתיהן בגודל n) לתוך 2 מערכים בהתאמה כך שלמערכים הללו נכנסים כל הנקודות בסטריפ,

כלומר כל הנקודות שנכנסות בקטע [median – width/2 , median + width/2] בהתאם לציר שהוכנס. פעולה זו לוקחת O(|B|) כיוון שאנחנו רצים מהחציון לשני הכיוונים ועוצרים בקצוות הקטע. לאחר מכן מתבצעת בדיקה הבודקת האם |B|log|B| גדול מ-n כאשר |B| הן כמות הנקודות שהוכנסו לסטריפ ו-n הן כמות הנקודות הקיימות במבנה הנתונים כרגע. אם n קטן מהשניים נבצע מיון של כל הנקודות לפי הציר הנגדי ע"י השיטה getPointsInRangeOppAxis שזמן הריצה שלה הוא O(n) אחרת נבצע מיון ע"י הפונקציה הסטטית Arrays.sort עם קומפרטור לציר הנגדי. זמן הריצה של פונקציה זו הוא O(|B|log|B|).

לאחר מכן, נרוץ על המערך אשר ממויין לפי הציר הנגדי ועבור כל נקודה נבדוק את המרחק שלה מ-7 הנקודות הבאות לה באמצעות לולאות for **(* הסבר גיאומטרי בהמשך *).** הבדיקה מתבצעת 7n פעמים כלומר בזמן O(|B|) ע"י הפונקציה distance שזמן הריצה שלה הוא O(1). ברגע שנמצא זוג נקודות הכי קרובות נכניס אותן למערך ולבסוף נחזיר את המערך עם הנקודות שמרחקן הוא הקטן ביותר.

לסיכום זמן הריצה נחלק למקרים:

- **O(|B|log|B|)** = O(|B|*1) + O(|B|log|B|) + O(|B|)**,** למקרה בו |B|log|B| קטן מ-n.
- **O(n)** = O(n*1) + O(n) + O(|B|), למקרה בו n קטן מ-|B|log|B|. כמו כן n תמיד גדול או שווה ל-|B|.

אנחנו בוחרים את המינימלי מבין שני המקרים ובסך הכל נקבל זמן ריצה של **O(min(|B|log|B|,n)**.

<u>הסבר גיאומטרי ל-(*):</u> התבססנו על המשפט הגיאומטרי הבא: במלבן בגודל d*2d קיימות לכל היותר 7 נקודות כך שמרחקן הוא לפחות d. ההוכחה מתבססת על כך שניתן לחלק את המלבן לשני ריבועים בגודל d*d ובגלל שהמרחק בין כל שני נקודות הוא לפחות d אז בכל ריבוע כזה יכולות להיכנס 4 נקודות סה"כ. כלומר כל נקודה במלבן ניתנת להשוואה עם לכל היותר 7 נקודות אחרות.

השיטה ()nearestPair:

השיטה : nearestPointsתחילה הפונקציה מעתיקה את 2 הרשימות המקושרות)שתיהן בגודל (n לתוך 2 מערכים בהתאמה, באמצעות לולאת for פעולה שלוקחת . (O(n הפונקציה מחשבת באמצעות הפונקציה getLargestAxisאת הציר הגדול יותר כדי לדעת עם איזה ציר נעבוד, פעולה שלוקחת .(O(1 לאחר מכן הפונקציה קוראת לפונקציה רקורסיבית nearestXאו nearestY(תלוי בציר הגדול יותר, 2 הפונקציות סימטריות). בפונקציה הרקורסיבית מתבצעת הקטנת המערכים בחצי בכל קריאה רקורסיבית עד תנאי העצירה ולכן ישנן (log(n קריאות רקורסיביות. בתוך הפונקציה הרקורסיבית ישנן קריאות לפונקציה subarrayשעוברת על כל המערך)בגודל (n פעם אחת ולכן לוקחת זמן ריצה ,(O(n לפונקציה yDivideו xDivide -שעוברות על המערך)בגודל (n פעמיים ולכן לוקחות זמן ריצה .(O(n כמו כן בפונקציה יש שני לולאות for כאשר כל אחת מהן עוברת על מערך בגודל n פעם אחת ולכן לוקחת זמן ריצה של .(O(n לבסוף יש לולאת for בתוך לולאת for כאשר לולאת ה for-הפנימית מתבצעת 7 פעמים לכל היותר ובתוכה מספר פעולות קבועות ולכן לוקחת זמן ריצה של (O(1 ולולאת ה for-החיצונית עוברת על מערך בגודל n פעם אחת ולכן לוקחת זמן ריצה של .(O(n בסה"כ קיבלנו שבתוך הפונקציה הרקורסיבית ישנן מספר פעולות שזמן הריצה שלהן הוא (O(nולכן זמן הריצה של כל קריאה רקורסיבית הוא ,(O(n כמו כן ישנן (log(n קריאות רקורסיביות ולכן זמן הריצה של הפונקציה הרקורסיבית הוא .((O(n*log(n אז בסה"כ ב nearestPoints -ישנן מספר פעולות ב (O(n-ועוד פעולה אחת ב((O(n*log(n -ולכן בסה"כ זמן הריצה של הפונקציה הוא <u>.((O(n*log(n</u>

חלק הבונוס:

6.5:

על מנת ליצור שני DT חדשים מ-DT קיים נבצע מספר פעולות:

- ניצור בנאי מעתיק המקבל DT.
- ניצור 2 DT'ים חדשים בעזרת הבנאי המעתיק: אחד יהיה לכל הנקודות הקטנות מהחציון (נקרא למבנה זה DT1) והשני יהיה לכל הנקודות הגדולות מהחציון כולל את החציון עצמו (נקרא למבנה זה DT2). תהליך זה לוקח זמן ריצה של (O(2n עבור כל מבנה נתונים, מפני שאנחנו רצים פעם אחת על הרשימה לפי ציר X ויוצרים רשימה חדשה במבנה הנתונים החדש ופעם שנייה רצים על הרשימה לפי Y ויוצרים אותה במבנה הנתונים החדש.
  סה"כ: (O(n.
- כעת נמחק מה-DT1 את כל האיברים הגדולים מהחציון (כולל אותו) באמצעות השיטה narrowRange, כנ"ל לגבי DT2 נמחק את כל האיברים הקטנים מהחציון. כבר יודעים שזמן הריצה של השיטה narrowRange הוא (O(n.

לכן עבור כל התהליך הנ"ל זמן הריצה הוא (O(n.

חלק ג':

הוספנו בונוס של סטים לעבודה.

אפשר להתייחס אל בעיית המטוסים כאל בעיית המחסור בפוקימונים בעולם בשל העובדה שפוקימוני אש ופוקימוני מים לא מסתדרים ביחד, חשמל עלולים לסכן את חיי פוקימוני המים ואדמה לא מסתדרים עם פוקימיני הרעל. כמובן אין צורך להזכיר את המחסור בפוקימונים אגדיים בכדור הארץ.

צירפנו לעבודה תמונה המתארת את המרחק המינימלי הנדרש לכל פוקימון (שאנחנו מכירים לפחות) מהפוקימונים שמסכנים את חייו. לדוגמה: צ'ריזארד יכול להיות במרחק יחסית קרוב לסקווירטל, אך רצוי שיהיה במרחק גדול יותר מבלסטויז.

אפשר להבחין שקטרפי וקקונה הם הפוקימונים שמסתדרים הכי טוב ביחד, לכן המרחק המינימלי ביניהם הוא הכי קטן.