מבני נתונים – משימה 2

מאור אסייג 318550746

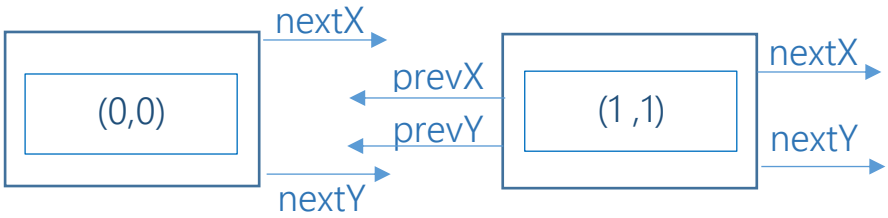
רפאל שטרית 204654891

המחלקה להנדסת חשמל ומחשבים, התכנית להנדסת מחשבים

מבני נתונים 202.1.1011

חלק ב'

1. בחרנו להשתמש במבנה נתונים הנעזר במשתנים מסוג Container אותם הגדרנו דמויי LinkList דו כיווני לכל ציר.
לכל אובייקט ישנם מצביעים - מצביע לחוליה הבאה והקודמת בציר x, ומצביע לחוליה הבאה והקודמת בציר y. בצורה זו אנו בונים "2 רשימות" ממוינות, כל רשימה בעלת n נקודות בעזרת n אובייקטים בלבד.
אובייקט קונטיינר ויזואלית :




```
public class Container{
    private Point data;//Don't delete or change this field;
    private Container prevX;
    private Container nextX;
    private Container prevY;
    private Container nextY;
```

במבנה הנתונים DatatStructure הגדרנו מספר שדות הכרחיים להשלמת המשימה :

```
public class DataStructure implements DT {
    private Container minx;
    private Container maxx;
    private Container miny;
    private Container maxy;
    private Container current; //current point
    private int size;
```

שדה current שימושי לחישוביים זמניים על מבנה הנתונים, כמו ריצה על האובייקטים וכדומה.

מימוש שיטות הממשק DT וניתוח זמן ריצה

```
void addPoint(Point point);
```

אלגוריתם :

1. במידה ומבנה הנתונים ריק, הגדר בהתאם את כל שדותיו שיצביעו לנקודה זו והגדל את size ל 1.
2. אחרת, נבנה קונטיינר חדש toAdd עם המידע של הנקודה point. נמצא את המיקום המתאים לנקודה זו **בציר x** (התחלה – minx וע"י לולאה להתקדם בעזרת current).
   - 2.1 הגדר את המצביעים לקונטיינר הקודם, לקונטיינר העוקב ולקונטיינר המכיל את הנקודה הנדרשת להוספה. בקונטיינר הנוכחי נשאיר באופן זמני את המצביעים הקשורים לציר ה y כ null.
3. מצא באופן זהה ל 2 את המיקום **בציר y.**
   - 3.1 באופן דומה ל 2.1 נגדיר את המצביעים הרלוונטיים לציר y.
   - 3.2 נגדיר את המצביעים של הקונטיינר toAdd לחוליות הרלוונטיות בציר y.

ניתוח זמן ריצה : במקרה הגרוע נצטרך לנוע על ציר x ועל ציר y על כל החוליות במבנה הנתונים עד שנגיע לקצה המבנה (כלומר הנקודה להוספה היא הנקודה עם ערכי y,x הגדולים ביותר) לכל נקודה שאנו עוברים אנו מבצעים O(1) פעולות – **בסה"כ O(n)** ✓.

Point[] getPointsInRangeRegAxis(**int** min, **int** max, Boolean axis);

אלגוריתם :

1. קריאה לפונקציית עזר NumInRangeAxis המקבלת את פרמטרי הפונקציה – השיטה מחזירה את מספר הנקודות ב DS עם הערך בציר axis שבטווח הרצוי.
2. אם אין נקודות בטווח, החזר מערך ריק.
3. על פי הציר הנדרש - נרוץ בלולאה על מבנה הנתונים מתחילתו ועד סופו ונוסיף את הנקודות הרלוונטיות לטווח על ידי בדיקה מתמטית. הלולאה תיעצר כשמספר הנקודות הידוע מראש הוכנס למערך המוחזר.

פונקציית עזר NumInRangeAxis : אלגוריתם זהה, מקבלת את פרמטרי הפונקציה ומחזירה מספר שלם המייצג את מספר הנקודות שבטווח. בהינתן ועברנו את הגבול המקסימלי הלולאה תיעצר (ייעול זמני ריצה).

ניתוח זמן ריצה : במקרה הגרוע הטווח מכיל את כל האובייקטים במבנה הנתונים, אזי פונקציית העזר תספור את כל הנקודות במבנה(=תעבור עליהן עם מספר סופי של פעולות O(1)) -> **O(n).** והשיטה גם תעבור על כל האובייקטים כשהיא מכניסה את המידע שלהן למערך ולכן גם היא O(n), **בסה"כ O(n)** ✓.

Point[] getPointsInRangeOppAxis(**int** min, **int** max, Boolean axis);

אלגוריתם :

לוגיקה זהה ל **getPointsInRangeOppAxis**, אך כעת ממיינת את הנקודות לפי הציר הנגדי (על ידי ריצה בציר axis! בשלב **3**).

ניתוח זמן ריצה : במקרה הגרוע הטווח מכיל את כל האובייקטים במבנה הנתונים, אזי פונקציית העזר תספור את כל הנקודות במבנה(=תעבור עליהן עם מספר סופי של פעולות O(1)) -> **O(n).** והשיטה גם תעבור על כל האובייקטים כשהיא מכניסה את המידע שלהן למערך ולכן גם היא O(n), **בסה"כ O(n)** ✓.

```
double getDensity();
```

1. מחשבת את צפיפות מבנה הנתונים על ידי הנוסחה $\frac{n}{(X_{max}-X_{min})(Y_{max}-Y_{min})}$.

ניתוח זמן ריצה : מספר סופי של, **בסה"כ O(1)** ✓.

```
void narrowRange(int min, int max, Boolean axis);
```

אלגוריתם :

1. נרוץ תחילה על הציר המבוקש החל מהחוליה minx (בהגב"כ), במידה וערכי הנקודה שבקונטיינר קטן מ min נכנס לתנאי.
   - **1.1** נגדיל את מספר הנקודות שאנו מוחקים ב 1.
   - **1.2** נדאג למחוק את המצביעים הקשורים לחוליה זו בציר הנגדי על ידי קריאה לפונקציית העזר narrowOppPoint.
   - **1.3** אם הגענו לקצה מבנה הנתונים, נעלה דגל.
2. הגענו לחוליה הראשונה שלא מקיימת את התנאי המוצג ב (**1**), נעדכן את המינימום החדש של מבנה הנתונים לחוליה הנוכחית.
3. כעת נרוץ אחורה החל מהחוליה המקסימילית maxx ונבדוק האם החוליה גדולה מהmax.
   - **3.1** נבצע את הלוגיקה המוצגת ב1.1 עד 1.3.
4. נחסיר מגודל המבנה את מספר הנקודות שמחקנו.

פונקציית עזר narrowOppPoint : עדכון המצביעים של החוליה הבאה בתור ושל החוליה הקודמת להתעלמות מהחוליה הנוכחית, ובכך אנו מוחקים אותה ממבנה הנתונים.

ניתוח זמן ריצה : פונקציית העזר מבצעת מספר סופי של פעולות-> O(1). השיטה במקרה הגרוע תבדוק את כל הנקודות במבנה הנתונים (כך שהתחום הנתון אינו חופף לערכים במבנה הנתונים) ותבצע בכל בדיקה שכזו מספר סופי של פעולות ולכן בסה"כ **O(n) worstcase**, באופן כללי השיטה תעבור על מספר הנקודות הרצויות למחיקה ובכל מעבר תבצע מספר סופי של פעולות ולכן זמן הריצה שלה **O(A)** כאשר **A** זה מספר הנקודות למחיקה ✓.

Boolean getLargestAxis();

אלגוריתם :

1. ביצוע חישוב מתמטי של רוחב הצירים על ידי ערכי maxx, minx, miny, maxy.
2. החזרה של התנאי widthX > widthY.

ניתוח זמן ריצה : השיטה מבצעת מספר סופי של פעולות ולכן **בסה"כ O(1)** ✓.

Container getMedian(Boolean axis);

אלגוריתם :

1. נחשב את כמות הפעמים שנצטרך לנוע בכדי להגיע לחציון ביחס לגודל המבנה **this.size/2**, על פי ההוראות.
2. על פי הציר הנדרש ננוע החל מ minx (בהגב"כ) כמספר הפעמים שחישבנו ב 1, ונחזיר את החציון.

ניתוח זמן ריצה : נכנס לתנאי n/2 פעמים, כל כניסה מלווה בקידום מצביע O(1) **ובסה"כ O(n)** ✓.

Point[] nearestPairInStrip(Container container, double width, Boolean axis);

אלגוריתם :

1. חישוב טווח הערכים של הרצועה לפי הציר הנדרש.
2. נספור את מספר הנקודות שברצועה ע"י פונקציית העזר NumInStripAxis, בעזרת האתחול המתאים (הכנסת ערכים למערך הנשלח לפונקציה – כיוון שהיא בעלת ייעוד כפול).
3. במידה וישנם יותר מ 2 נקודות ברצועה, נכניסם למערך בגודל המתאים בעזרת NumInStripAxis ואתחול מתאים של המערך הנשלח אליה.
4. נותר למיין את המערך על פי הציר הנגדי. ישנם שתי גישות, נבדוק איזו מהן היא המינימלית ונמיין לפיה.
כאשר B מספר הנקודות ברצועה : $n \text{ or } Blog_2B$

4.1 $Blog_2B$: נמיין לפי Arrays.sort

4.2 n : נחשב את המינימום והמקסימום שמופיעים במערך (בעת ספירת הנקודות דאגנו לשמור את האינדקס שיכיל את הנקודה המינימלית לפי ערכי הציר המתאים במשתנה minIndex, המקסימום הינו האיבר האחרון שהוכנס למערך (בהכנסה התחלנו מ i=arr.length-1, לכן האיבר המקסימלי יהיה באינדקס 0)).

4.2.1 נקרא לפונקציה getPointsInRangeOppAxis עם תחום הערכים שחישבנו והציר המתאים, נקבל בחזרה מערך ממוין של הנקודות.

5. נמצא את זוג הנקודות הקרוב ביותר בעזרת פונקציית העזר nearestPointInArray שבודקת כל נקודה עם 7 הנקודות אחריה (תחת ההנחות שהונחו בשאלה, ההוכחה תובא בהמשך).

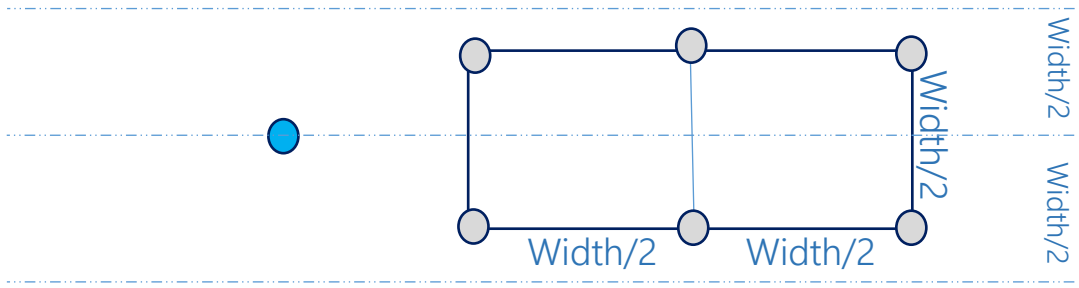
6. נחזיר מערך המכיל את זוג הנקודות עם המרחק המינימלי ביותר ברצועה.

פונקציית עזר Distance : מקבלת 2 נקודות ומחשבת את המרחק בינהן.

המרחק בין שתי נקודות $(X_1, Y_1), (X_2, Y_2)$ הינו $\sqrt[2]{(X_1 - X_2)^2 + (Y_1 - Y_2)^2}$

פונקציית עזר NumInStripAxis : ייעודה כפול, בעזרת אתחול מתאים של מערך strip נוכל לספור את הנקודות ברצועה התחומה בין min ל max בציר axis (לוגיקה זהה ל getPointsInRangeRegAxis), ובעזרת אתחול אחר נוכל להכניס את הנקודות המתאימות למערך strip.

פונקציית עזר nearestPointInArray : בודקת במערך ממוין לפי הציר הנגדי את 7 הנקודות הבאות אחריה, על פי הוכחה מתמטית (ההנחה היא שהמרחק המינימלי בין כל נקודה ונקודה הינו width/2)



המשך ->

ניתוח זמן ריצה : נסמן B מספר הנקודות ברצועה.

השיטה NumInStripAxis רצה מתחילת מבנה הנתונים עד שמגיעים לנקודה שמחוץ לטווח המינימום, באופן זהה מסוף מבנה הנתונים לגבי המקסימום – לכל נקודה שבטווח מתבצעים מספר סופי של פעולות ולכן -> O(B).
שלב 4 ממין את המערך בזמן הריצה המינימלי $n\ or\ Blog_2B$.
שלב 5 לכל נקודה במערך עם B נקודות מתבצעות 7 השוואות, ולכן $O(1) + 7O(B) = O(B)$

ולכן בסה"כ

$O(1) + O(B) + O(\min[n, Blog_2B]) = O(\min[n, Blog_2B])$.✓

.2 Point[] nearestPair();

אלגוריתם : כפי שמופיע בהדרכה, שיטת Divide and Conquer.
פונקציית עזר nearestPairRec מבצעת בצורה רקורסיבית את שלבים 1 עד 6 בהדרכה.

1. אם יש רק זוג נקודות, החזר אותם. אם יש פחות, החזר זוג ריק (או null).
2. מוצאת את הציר הגדול ביותר axis נניח בלי הגבלת הכלליות ציר X
3. מוצאת את החציון median בציר X
4. מחשבת רקורסיבית את הזוג הקרוב ביותר עבור כל הנקודות הגדולות מ median (כולל) (נקודות שקורדינטת ה-X שלהם גדולה מקורדינטת ה-X של ה- median) ורקורסיבית את הזוג הקרוב ביותר עבור כל הנקודות הקטנות מ median לפי ציר X (נקודות שקורדינטת ה-X שלהם קטנה מקורדינטת ה-X של ה- median)
5. בוחרת את הזוג הקרוב יותר מבין שתי הזוגות בצעד 4 ומחשבת את המרחק ביניהן minDist
6. בודקת האם ברצועה (בציר X) ברוחב 2*minDist אשר האמצע שלה זה ערך ה X של הנקודה median, יש זוג נקודות שמרחקן קטן מ minDist
   a. אם קיימות זוג נקודות כאלו אזי הפונקציה מחזירה את הזוג הזה
   b. אחרת, הפונקציה מחזירה את הזוג מצעד 5

תנאי העצירה של הפונקציה הרקורסיבית : מערך בגודל 3 נקודות לכל היותר, נקרא לפונקציה nearestPair3Points המחשבת את המרחק הקצר ביותר מבין 3 הנקודות הללו (לפחות 2 נקודות, אחרת החזר null, מספר סופי של פעולות).

ניתוח זמן ריצה : (ראשית מובא הסבר על המימוש)

nearestPairRec $(Point[\quad] \; range, boolean\ axis)$ $cut\ in\ half\ the\ array, if\ we\ get\ into\ the\ stop\ condition(at\ most\ 3\ point\ in\ range\ array)\ then\ it\ take\ \theta(1)$
$to\ calculate\ the$ min $distance\ between\ thus\ (at\ most\ 3) points.$

nearestPair

$step\ 1: if\ we\ have\ in\ the\ DS\ only\ 2\ points\ returen\ them, if\ less\ return\ null. O(1)$
$step\ 2: find\ largest\ axis\ by\ the\ minx. maxx, miny, maxy\ pointers\ in\ the\ DS. O(1)$
$step\ 3: find\ median\ by\ runnig\ from$ min $axis\ pointer\ to\ \frac{this.size}{2}\ of\ the\ DS. O(n)$
$step\ 4: find\ the\ nearest\ points\ recursively\ in\ the\ (min\ to\ median)\ d1\ and\ in\ (median\ to$ max$)\ d2, then\ find\ the$ min $between\ thus\ 2\ distances.\ each\ side\ use\ getPointsInRangeRegAxis\ to\ build\ an\ sorted\ array$
$step\ 5: find\ the$ min $of\ (d1, d2). O(1)$
$step\ 6: calculate\ the\ nearest\ point\ in\ the\ strip\ width$
$= minDistance * 2$
$around\ the\ median. we\ find\ the\ bounders, get\ the\ points\ into\ the\ array(sorted\ by\ the\ opp\ axis. O(n)$

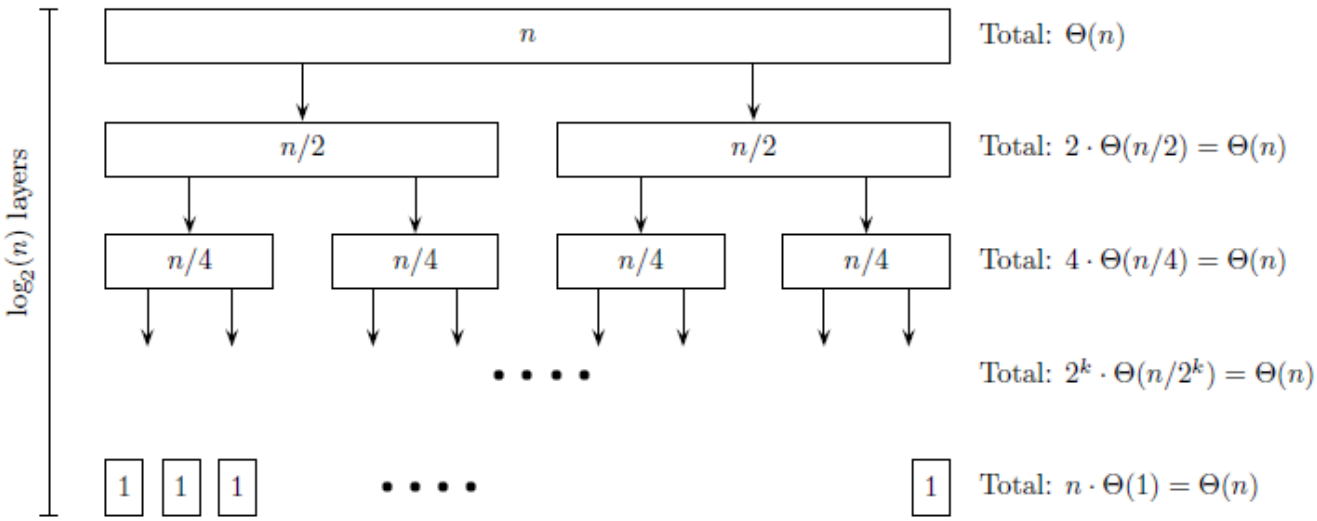
Runtime Analysis :

$$T(n) = T\left(\frac{n}{2}\right) + T\left(\frac{n}{2}\right) + O(n) + O(n)$$
$$= T\left(\frac{n}{4}\right) + T\left(\frac{n}{4}\right) + T\left(\frac{n}{4}\right) + T\left(\frac{n}{4}\right) + 3O(n)$$
$$= \cdots = \sum_{k=1}^{k=i} T(<3) + iO(n) = O(\log_2 n) + O(nlog_2 n)$$

$we\ found\ i\ by\ (we\ choose\ 1\ to\ easy\ the\ calculation):$

$$\frac{n}{2^i} = 1 \rightarrow 2^i = n \rightarrow i = \log_2 n.$$

$Thus\ altogheter\ T(n) = O(nlog_2 n)$ ✓.



.3

Pseudo code, Split :

$= is\ the\ counter \rightarrow$

```
arr [] split (int value, boolean axis)
    arr [] ans = arr[4]//O(1)
    if(axis)
        low_value = first_X        //O(1)
        high_value = max_X         //O(1)
        ans [1] = low_median       //O(1)
        ans[4] = high_median       //O(1)
        while(low_value.get_x < value AND value ≤ high_value.get_x)
                                   //low_value = low_value.next
            high_value = high_value.prev
        if(low_value.get_x < value)//O(1)
            ans[2] = high_value//O(1)
            ans[3] = high_value.next//O(1)
        else
            ans[2] = low_value.prev
            ans[3] = low_value
    else   //same run time
        low_value = first_Y
        high_value = max_Y
        ans [1] = low_median
        ans[4] = high_median
        while(low_value.get_Y < value AND value ≤ high_value.get_Y)
            low_value = low_value.next
            high_value = high_value.prev
        if(low_value.get_Y < value)
            ans[2] = high_value
            ans[3] = high_value.next
        else
            ans[2] = low_value.prev
            ans[3] = low_value
    return ans
end
```

המשך ->

אלגוריתם :

- הפונקציה split מקבלת value ו axis , תפקידה הוא להחזיר 2 אוספים של נקודות ,אוסף אחד ממינימום ועד value ואוסף שני מvalue ועד המקסימום.
- הפונקציה split מחזירה מערך ans המכיל את תחומי האוספים ,כלומר האוסף הראשון (הקטן מvalue) תחום בans[1] ועד ans[2] . האיברים שבמערך האלה הם מצביעים ל container שהוא איבר במבנה הנותנים הקיים. בעצם על ידי ניצול מבנה הנתונים הקיים אפשר לעבור איבר-איבר ברשימה על ידי המעבר בין ה containers.
- בהתאמה ans[3] ו ans[4] מכילים את התחום השני וגם כאן אילו מצביעים על containers מהמבנה הקיים.
- המעבר על האיברים מתבצע על ידי המעבר בתוך מבנה הנתונים שבנינו כיוון שפונקציית Split מחזירה מצביעים לאיברים קיימים ושמקושרים ביניהם.

ניתוח זמן ריצה :

בפונקציה שבנינו ישנם 2 if שמחליטים באיזה ציר אנחנו , בכל אחד מה if יש לולאת while . הלולאה מתחילה להתקדם במקביל גם מהמקסימום וגם מהמינימום של המבנה נתונים עד אשר תנאי הלולאה לא מתקיים . בשיטה זאת אנו נמשיך להיות בלולאה עד אשר המצביעים לאוסף הקטן יותר ימצאו מה שאפשר לזמן הריצה של פונקציה זאת להיות **O(|C|)** . בנוסף אפשר לראות את ניתוח הזמן בפסאודו קוד.

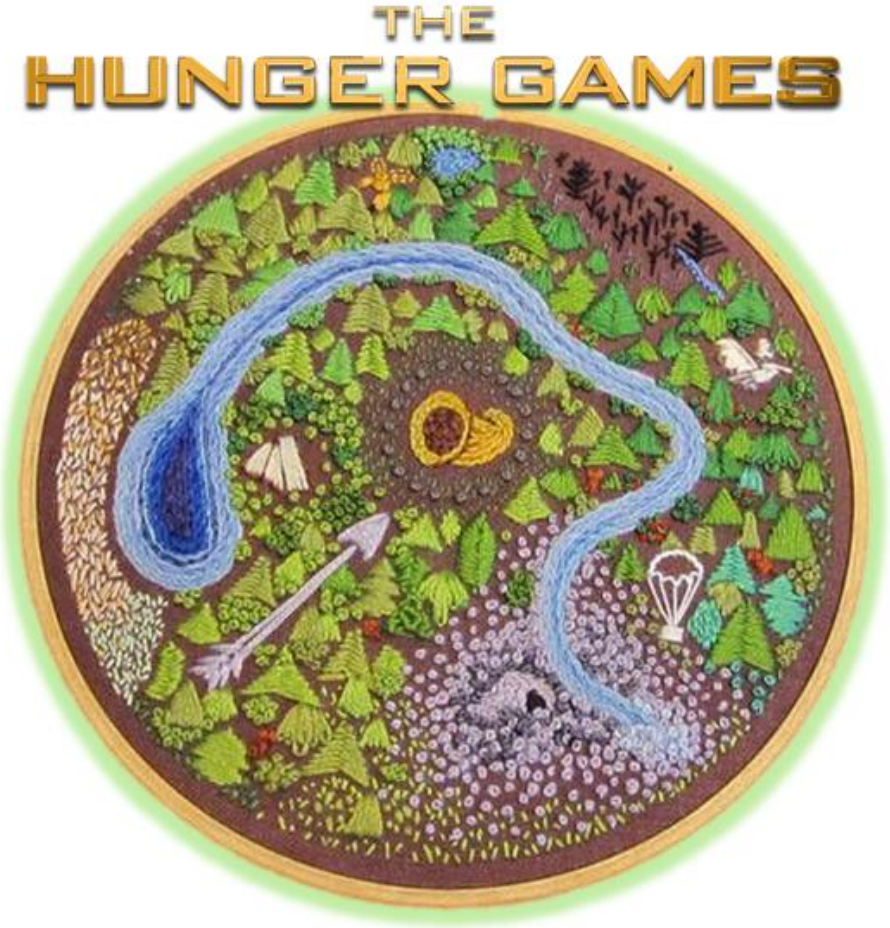
$$
\begin{aligned}
\mathrm{T}(n) &= O(1) + O(1) + O(1) + O(1) + O(1) + O(1) \\
&\quad + O(1) + O(1) + O(|C|) \\
&= O(1) + O(|C|) \\
&= O(|C|) \; when\ C\ is\ the\ numer\ of\ points\ in\ the\ smallest\ collection.
\end{aligned}
$$

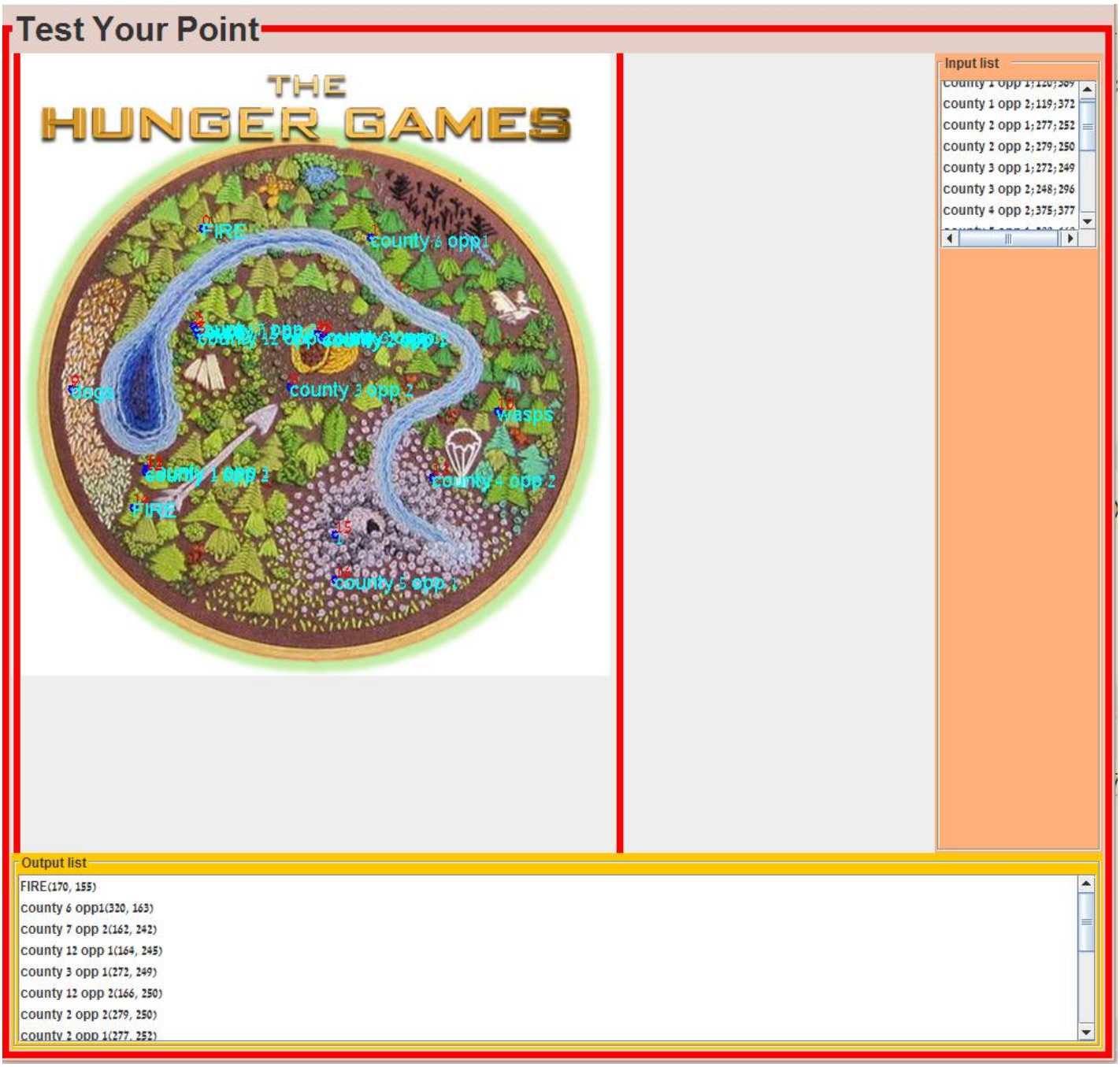
חלק ג' בונוס

קובץ ראשון :

מערכת המתארת את מיקום המשתתפים במשחקי הרעב, ואירועים נוספים (כמו שריפה, צרעות משוחררות וכדומה).

נרצה לדעת היכן ממוקמים המשתתפים ומיהם המשתתפים הקרובים ביותר (הנמצאים בצוות או שעומדים להיתקל אחד עם השני).



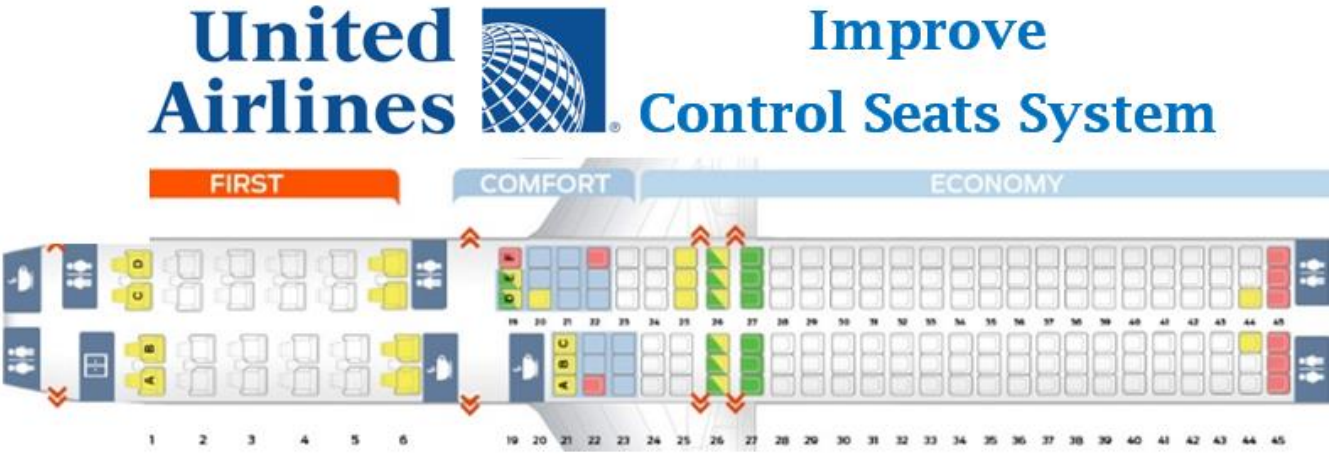

Test Your Point
THE HUNGER GAMES
FIRE
county 6 opp1
county 3 opp 2
wasps
county 1 opp 2
county 4 opp 2
FIRE
county 5 opp 1
Input list
county 1 opp 2;119;372
county 2 opp 1;277;252
county 2 opp 2;279;250
county 3 opp 1;272;249
county 3 opp 2;248;296
county 4 opp 2;375;377
Output list
FIRE(170, 155)
county 6 opp1(320, 163)
county 7 opp 2(162, 242)
county 12 opp 1(164, 245)
county 3 opp 1(272, 249)
county 12 opp 2(166, 250)
county 2 opp 2(279, 250)
county 2 opp 1(277, 252)

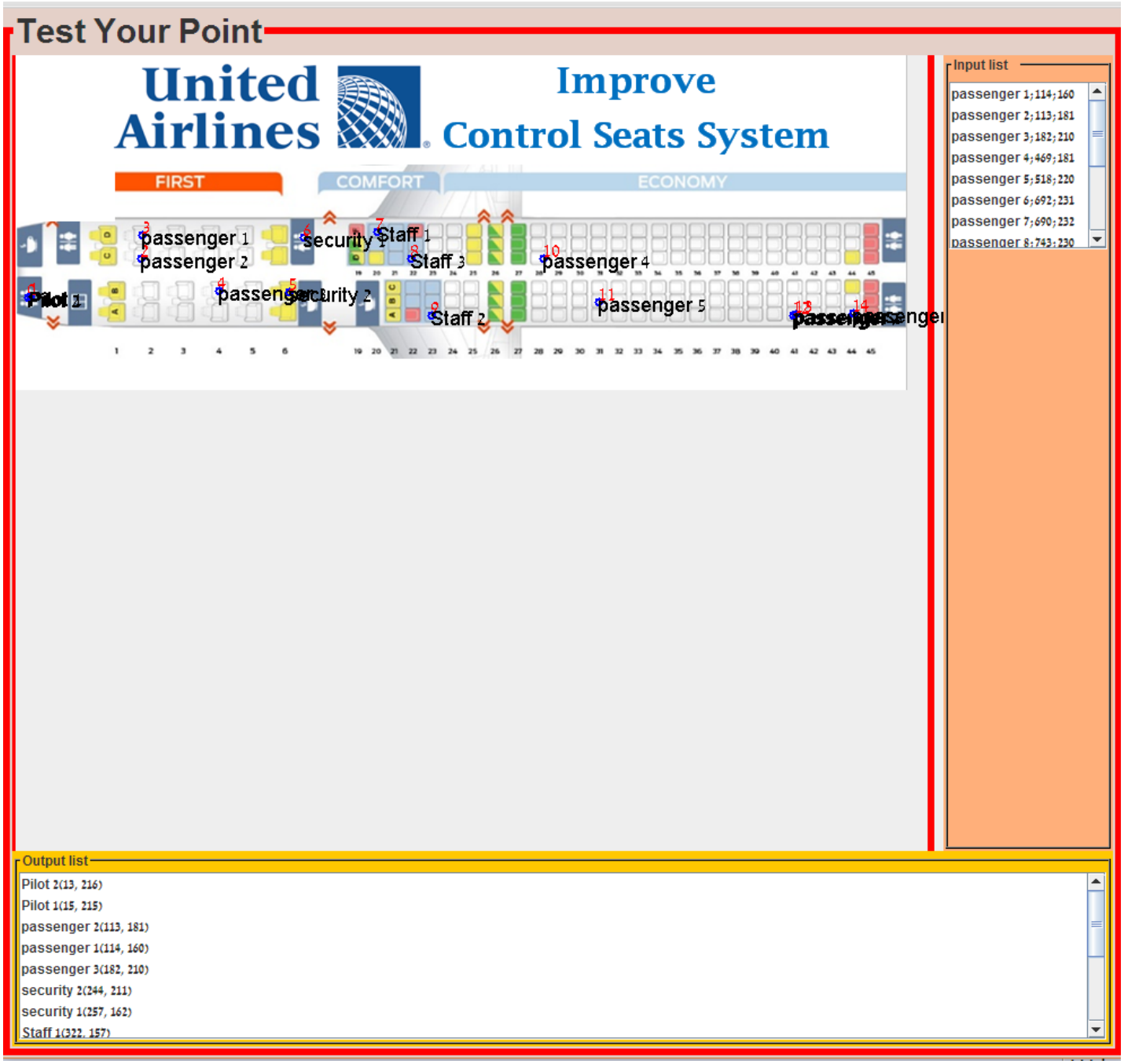
קובץ שני :

נרצה לדעת אלו מקומות ישיבה תפוסים על ידי נוסעים, אנשי צוות, מאבטחים וטייסים.

מבנה הנתונים שלנו כמובן **לא תומך בהסרת נוסעים**, בהתאם לנוהל הנסיעות שלנו - על כן אין פונקציית Remove במימוש.

בכדי להימנע מחשיפה תקשורתית מוטעית איננו מסירים לעולם נוסעים ממושבם, אנחנו פשוט יוצרים מבנה נתונים חדש המדמה לכל טיסה את מקומות הישיבה במטוס.





Until next time, thank you.