דו"ח פרויקט

אלדד ספפורטס, גיא קומש, עופר חביב, ארי פטל, ברק קויפמן, ספיר צוארו, יניר עטר

הגדרת הבעיה:

קלט: אוסף נקודות במרחב, כאשר כל נקודה מיוצגת על ידי שתי קואורדינטות (x, y).

הגדרות:

שתי נקודות p_1, p_2 ניתנות להשוואה אם אחת גדולה מהאחרת בשתי הקואורדינטות.
כלומר עבור הנקודות $p_1 = (x_1, y_1)\,, p_2 = (x_2, y_2)$ מתקיים ש $p_1 < p_2$ אם ורק אם $x_1 < x_2$ וגם $y_1 < y_2$.

שרשרת של נקודות היא סדרה של נקודות כך שכל נקודה ניתנת להשוואה לנקודה הבאה בסדרה וקטנה ממנה לפי ההגדרה. כלומר: $(p_1, p_2, p_3, \dots, p_n)$ כאשר $p_1 < p_2 < \cdots < p_n$.

משקל של שרשרת הוא סכום משקלי הנקודות שחברות בשרשרת, כלומר:

$$Weight\big((p_1, p_2, p_3, \dots, p_n)\big) = \sum_{i=1}^{n} Weight\ of\ p_i$$

משקל של סכום ריבועי משקלי הנקודות:

$$Weight\big((p_1, p_2, p_3, \dots, p_n)\big) = \sum_{i=1}^{n} (Weight\ of\ p_i)^2$$

משקל שרשרת מקסימלית באוסף – יסומן ב-W היא משקלה של השרשרת הכבדה ביותר הנמצאת באוסף הנקודות, כלומר:

$$W = \max_{for\ chain\ in\ collection} Weight(chain)$$

בתחילת הריצה לכל נקודה יש משקל התחלתי ששווה ל-1, לכן משקלה של כל שרשרת תהיה שווה לכמות הנקודות הנמצאות בה, לכן W היא אורך השרשרת הארוכה ביותר באוסף.

הבעיה:

מטרתנו היא תחילה למצוא את משקלה של השרשרת המקסימלית בגרף W, ולנסות להעלות באופן סלקטיבי את המשקלים של הנקודות באוסף שקיבלנו, עד כמה שניתן, מבלי לפגוע במשקל השרשרת המקסימלית בגרף W, כלומר שלא תתיכן שרשרת שמשקלה W' כך ש $W < W'$.

פלט: אוסף הנקודות עם המשקלים המעודכנים, עבור בעיית סכום המשקלים הרגילה ובעיית סכום ריבועי המשקלים.

צורת החשיבה:

מציאת W: תחילה חשבנו כיצד נוכל למצוא את משקל השרשרת המקסימלית באוסף (W) בצורה יעילה. תכננו אלגוריתם בעזרת תכנון דינמי שמקבל את הנקודות לאחר מיונן ומתחזק מערך שגודלו כמספר הנקודות באוסף, כך שכל תא הוא משקלה של שרשרת הכבדה ביותר המתחילה בנקודה. האלגוריתם מעדכן בצורה איטרטיבית עבור כל נקודה את משקל השרשרת הכבדה ביותר המתחילה בה, ובסוף מחזיר את המקסימום על המערך, שהוא W.

עדכון משקלים: לאחר שמצאנו את W, ניסנו להבין איזה משקלים של נקודות באוסף נוכל להגדיל מבלי להגדיל את W, ואיך לעשות זאת בצורה יעילה ובאיזה מבני נתונים כדאי לנו להשתמש כדי לקבל פתרון יעיל. במקרה של העלאת משקלה של נקודה, רצינו להבין אילו נקודות נצטרך לבחון ולא להגדיל את משקלן וזאת על מנת לא להגדיל את W. הגענו לפתרון שבו עבור כל נקודה, נמצא את משקל השרשראות הכבדות ביותר המתחילות בה, בדומה לאלגוריתם למציאת W, אך גם נשמור את **כל הנקודות שנמצאות על אותן השרשראות**. לדוגמא – אם מצאנו שמשקלן הכבד ביותר של שרשראות המתחילות בנקודה p_i הוא w_i, נשמור את כל הנקודות שנמצאות על כל שרשרת במשקל w_i המתחילה בנקודה p_i , ללא כפילויות. זאת מתוך ההבחנה כי בהגדלת משקלה של p_i, אלו הן הנקודות אותן נרצה לבחון בזהירות, ונצטרך לוודא שמשקל כל השרשראות העוברות בנקודת אלה לא עלה על W. צורת החשיבה בפתרון הבעיה הייתה לפעול באיטרציות, תוך ביצוע שני חישובים מרכזיים בכל איטרציה. החישוב הראשון הוא למצוא עבור כל נקודה, את אוסף הנקודות שנמצאות על השרשראות הכבדות ביותר המתחילות בה ומשקלן של אותן השרשראות, ולשמור את המיפוי מהנקודה לנתונים אלו ב$hash\ map$. החישוב השני מתבצע לאחר הראשון, ומקבל את ה$hash\ map$ שנבנה בראשון כארגומנט, רץ על אוסף הנקודות הממוין מהנקודה הקטנה ביותר (הקואורדינטות הקטנות ביותר) לגדולה ביותר ומדלג על נקודות לא רלוונטיות. עבור כל נקודה p_i, האלגוריתם בודק האם יכול להעלות את משקלה (אם משקל השרשראות הכבדות ביותר המתחילות בה קטן מW) ואם כן, מגדיל את משקלה ומוסיף אותה ואת כל הנקודות שמופו לנקודה מה- $hash\ map$, לרשימת הנקודות שנרצה לדלג עליהן בהמשך אותה איטרציה. מנגנון זה מבטיח שבעת הגדלת משקלה של נקודה ב-1, משקל כל השרשראות הכבדות ביותר המתחילות בה יוגדלו ב-1 בלבד – משקל שאר הנקודות על השרשראות האלו לא יוגדל. ביצוע שני החישובים האלו באיטרציה אחד אחרי השני, מבטיח שנגדיל את משקל הנקודות בצורה סלקטיבית, בכל איטרציה, תוך שמירה על W.

פרטי מימוש - הקוד:

<u>נכתב בPython.</u>

<u>עיבוד קובץ הדאטה:</u>

לשם חילוץ אוסף הנקודות מתוך גיליונות קובץ האקסל, השתמשנו בחבילה pandas.

1. $get\_sheets\_name()$ - מחזירה את שמות הגיליונות בקובץ.
2. $get\_points\_from\_sheet(sheet\_name)$ מחזירה את אוסף הנקודות מגיליון כלשהו.

<u>פתרון הבעיה:</u>

מבני נתונים:

1. $weights\_map$ - hash map הממפה בין נקודה לבין משקלה, כלומר:
$$weights\_map[p_i] = weight\ of\ p_i$$
2. $point\_set\_map$ - hash map הממפה בין נקודה לבין זוג ערכים - משקל השרשרת הכבדה ביותר המתחילה בנקודה ואוסף הנקודות (set של Python) הנמצאות על כל השרשראות הכבדות ביותר המתחילות באותה הנקודה, כלומר:
$$point\_set\_map[p_i] = (w_i, set\ of\ all\ points\ on\ w_i\ weighted\ chains\ starting\ at\ p_i)$$
* נשים לב כי w_i הוא משקל השרשרת הכבדה ביותר המתחילה בp_i, ולא משקל הנקודה עצמה.
3. $points$ – מערך המייצג את אוסף הנקודות שחולץ מהגיליון. הנקודות במערך ממוינות לפי קואורדינטת הx, ואם שוות אז לפי y, מהקטנה לגדולה. המערך אינו מכיל את משקל הנקודות.

פונקציות החישוב:

1. $get\_W(points, weights\_map, squared)$ - אלגוריתם תכנון דינמי לחישוב W. <u>קלט:</u> מערך הנקודות הממוין ומשקל הנקודות, כאשר בתחילה משקל כל נקודה הוא 1, ודגל המצביע האם לחשב עבור בעיית סכום ריבועי המשקלים. האלג' רץ על מערך הנקודות הממוין מהסוף להתחלה (כלומר מתחיל בנקודה הכי גדולה באוסף ומסיים בקטנה ביותר). עבור הנקודה p_i האלג' מחשב את משקל השרשרת המקסימלית המתחילה בה $dp[i]$, על ידי מעבר על כל קבוצת כל הנקודות p_j שניתנות להשוואה איתה וגדולות ממנה ($Relevant(p_i)$), ומציאת הנקודה p_j שמשקל השרשרת המקסימלית המתחילה בה הוא הגדול ביותר, ועדכון הערך עבור הנקודה הנוכחית p_i, כלומר:
$$Relevant(p_i) = \{\, p_j \mid p_j > p_i \}$$
עבור סכום המשקלים הרגיל : $dp[i] = \max_{p_j \in Relevant(p_i)} (dp[j]) + weights\_map[p_i]$

עבור סכום ריבועי המשקלים: $dp[i] = \max_{p_j \in Relevant(p_i)}(dp[j]) + (weights\_map[p_i])^2$

כאשר הביטוי $\max_{p_j \in Relevant(p_i)}(dp[j])$ מייצג את משקל השרשרת הכבדה ביותר המתחילה בנקודה $p_j \in Relevant(p_i)$ כלשהי.

<u>פלט:</u> האלגוריתם יחזיר את הערך המקסימלי במערך dp - משקל השרשרת הכבדה ביותר באוסף – W, כלומר: $W = \max_{0 \le i < n}(dp[i])$.

2. $\boldsymbol{get\_set\_of\_points\_on\_heaviest\_chains(points, weights\_map)}$ - אלגוריתם הבונה את מבנה הנתונים $point\_set\_map$. עבור כל נקודה נמצא את אוסף הנקודות הנמצאות על השרשראות הכבדות ביותר המתחילות בה, ואת משקל השרשרת. הפונקציה מחזירה את מבנה הנתונים $point\_set\_map$ שבנתה.
   i. $iset$ – סט של אוסף הנקודות שנמצאות על השרשראות הכבדות ביותר המתחילות בנקודה p_i.
   ii. w_i – משקלן של השרשראות הכבדות ביותר המתחילות בנקודה p_i.

   המטרה של הפונקציה היא לבנות את $point\_set\_map$, כך שימפה בין p_i ל$iset$ ול-w_i.

פסודו קוד - $\boldsymbol{get\_set\_of\_points\_on\_heaviest\_chains}$:

$\boldsymbol{get\_set\_of\_points\_on\_heaviest\_chains}(points, weights\_map)$:
$\boldsymbol{Init}$:
$\boldsymbol{point\_set\_map} = empty\ hash\ map$
$\boldsymbol{n} \leftarrow length(points)$
$\boldsymbol{For}\ i\ from\ n-1\ to\ 0$:
$\quad iset = empty\ set$
$\quad w_i = 0$
$\quad \boldsymbol{For}\ j\ from\ i+1\ to\ n-1$:
$\quad\quad \boldsymbol{If}\ p_j > p_i\ and\ w_j > w_i$:
$\quad\quad\quad w_i = w_j$
$\quad \boldsymbol{For}\ j\ from\ i+1\ to\ n-1$:
$\quad\quad w_j, jset = point\_set\_map[p_j]$
$\quad\quad \boldsymbol{If}\ p_j\ not\ in\ iset\ and\ p_j > p_i\ and\ w_j = w_i$:
$\quad\quad\quad iset = iset \cup jset$
$\quad iset = iset \cup p_i$
$\quad \boldsymbol{If}\ iset\ size\ is\ greater\ than\ 1$:
$\quad\quad point\_set\_map[p_i] = (weight\_map[p_i], iset)$
$\quad \boldsymbol{Else}$:
$\quad\quad point\_set\_map[p_i] = (weight\_map[p_i] + w_i, iset)$
$\boldsymbol{Return}\ point\_set\_map$

*** קיימת פונקציה תאומה עבור פתרון בעיית סכום ריבועי המשקלים.**

3. $\boldsymbol{update\_points\_weights(points, point\_set\_map, weights\_map)}$ – אלגוריתם הרץ על מערך הנקודות הממוינות מתחילתו לסופו- מהנקודה הקטנה ביותר לגדולה ביותר. האלגוריתם מתחזק אוסף של נקודות $visited$ עליהן ידלג ולא ינסה להגדיל את משקלן. עבור כל נקודה, **אם היא לא נמצאת ב$\boldsymbol{visited}$**, האלגוריתם ימצא את משקל השרשראות הכבדות ביותר המתחילות בנקודה p_i w_i , ואת כל הנקודות על שרשראות אלו (יתכנו מס' רב כאלו) בעזרת $point\_set\_map$, ויכניס את הנקודות לאוסף $visited$. עתה נבדוק האם אפשר להגדיל את משקל הנקודה, כלומר אם $w_i < W$. ואם כן יגדיל את משקלה של הנקודה ב1 ב$weights_map$ ואת משקל השרשרת המקסימלית המתחילה בה ב1 ב$point_set_map$. עתה, מכיוון שהנקודות על השרשראות הכבדות המתחילות בה הוספו ל$visited$, האלגוריתם ידלג עליהן בהמשך ריצתו באיטרציה הנוכחית, ייתכן שנוכל להגדיל את הנקודה שוב באיטרציה הבאה. הפונקציה לא מחזירה ערך, אלא רק מעדכנת את משקל השרשראות והנקודות במבני הנתונים לאחר עדכון משקלה של נקודה.
*** קיימת פונקציה תאומה עבור פתרון בעיית סכום ריבועי המשקלים.**

פונקציות הרצת פתרון הבעיה:

1. $\boldsymbol{maximum\_weight\_iteration(points, weights\_map, squared)}$ - מריצה איטרציה בודדת לפתרון הבעיה. במידה ו- $squared = True$: תריץ איטרציה לבעית סכום ריבועי המשקלים. אחרת תריץ איטרציה לבעיית סכום המשקלים הרגילה.
הפונקציה תריץ תחילה את $get\_set\_of\_points\_on\_heaviest\_chains()$ לקבלת מבנה הנתונים $point\_set\_map$, ולאחר מכן תריץ $update\_points\_weights()$ עם מבנה הנתונים שקיבלה.

2. $\boldsymbol{run(points, squared)}$ – פונקציה הרצת פתרון הבעיה שנקראת מה-Main. הפונקציה מקבלת את מערך הנקודות הממוין, ודגל True / False המסמן האם לפתור את בעיית סכום ריבועי המשקולות או את הבעיה הרגילה. הפונקציה תריץ בכל פעם איטרציה - $\boldsymbol{maximum\_weight\_iteration()}$ עד אשר לא הצלחנו להגדיל משקל של אף נקודה מהאוסף באיטרציה האחרונה, או עד שנגמר הזמן שהוקצב מראש לריצה – MAX_RUNTIME_MINUTES (בברירת מחדל דקה). לאחר כל איטרציה, הפונקציה תבדוק האם נגמר הזמן שהוקצב מראש, ואם כך ,לא תבצע עוד איטרציה ותסיים. הפונקציה מחזירה מערך של הנקודות והמשקולות החדשים בגרף.

<u>הכנת קובץ הפלט:</u>

לשם הכנת הקובץ נעזרנו בחבילה openpyxl. הפלט של התוכנית הוא קובץ אקסל בשם ‘output.xlsx’ בו כל גיליון תואם לגיליון מקובץ הקלט ומכיל ארבעה עמודות: שתי עמודות עבור הקואורדינטות של הנקודות- "X" , "Y" , עמודה עבור המשקלים המעודכנים עבור כל נקודה מפתרון הבעיה לסכום המשקלים הרגיל – "W" ועמודה נוספת עבור פתרון הבעיה עבור סכום ריבועי המשקלים – "SQ".

1. $\boldsymbol{write\_output}(\boldsymbol{points}, \boldsymbol{weighted\_points\_map},\ \boldsymbol{squared\_map},\ \boldsymbol{sheet})$ - מקבלת את אוסף הנקודות, מיפויי הנקודות למשקליהן המעודכנים, ושם הגיליון הרלוונטי, וכותבת לקובץ האקסל בגיליון המתאים את הנקודות ומשקלן המעודכן

דוגמאות הרצה:

נריץ עבור הגיליון הראשון square_10000_samples_V0, עם מגבלת הרצה של ~דקה:

ריצה עבור סכום משקלים רגיל

חישוב W

איטרציות הרצה

זמן ריצה הכללי

עדכון המשקלים שבוצע

בדיקת נכונות

ריצה עבור סכום ריבועי המשקלים

```
$ py main.py
Reading data from Sheet: square_10000_samples_V0
---------Running with sum of regular weights...
Calculating W =  203
running iteration [1]
1. get_set_of_points_on_heaviest_chains()
10.0% 20.0% 30.0% 40.0% 50.0% 60.0% 70.0% 80.0% 90.0% 100.0%
2. update_points_weights()
time_passed: 19 sec
running iteration [2]
1. get_set_of_points_on_heaviest_chains()
10.0% 20.0% 30.0% 40.0% 50.0% 60.0% 70.0% 80.0% 90.0% 100.0%
2. update_points_weights()
time_passed: 36 sec
running iteration [3]
1. get_set_of_points_on_heaviest_chains()
10.0% 20.0% 30.0% 40.0% 50.0% 60.0% 70.0% 80.0% 90.0% 100.0%
2. update_points_weights()
time_passed: 52 sec
running iteration [4]
1. get_set_of_points_on_heaviest_chains()
10.0% 20.0% 30.0% 40.0% 50.0% 60.0% 70.0% 80.0% 90.0% 100.0%
2. update_points_weights()
time_passed: 69 sec
Iterations Completed!
running time : 71.8139283657074
In 4 iterations: total weight increase: 10000 -> 12746
Max chain weight (W) sanity check : first iteration = 203, after 4 iterations = 203
---------Running with sum of squared weights..
Calculating W =  203
running iteration [1]
1. get_set_of_points_on_heaviest_chains_squared()
10.0% 20.0% 30.0% 40.0% 50.0% 60.0% 70.0% 80.0% 90.0% 100.0%
2. update_points_weights_squared()
time_passed: 21 sec
running iteration [2]
1. get_set_of_points_on_heaviest_chains_squared()
10.0% 20.0% 30.0% 40.0% 50.0% 60.0% 70.0% 80.0% 90.0% 100.0%
2. update_points_weights_squared()
time_passed: 39 sec
running iteration [3]
1. get_set_of_points_on_heaviest_chains_squared()
10.0% 20.0% 30.0% 40.0% 50.0% 60.0% 70.0% 80.0% 90.0% 100.0%
2. update_points_weights_squared()
time_passed: 57 sec
running iteration [4]
1. get_set_of_points_on_heaviest_chains_squared()
10.0% 20.0% 30.0% 40.0% 50.0% 60.0% 70.0% 80.0% 90.0% 100.0%
2. update_points_weights_squared()
time_passed: 75 sec
Iterations Completed!
running time : 78.71846652030945
In 4 iterations: total weight increase: 10000 -> 10124
Max chain weight (W) sanity check : first iteration = 203, after 4 iterations = 203
```

כיצד להריץ את הפרוייקט:

1. יש לוודא כי מותקן פייתון (השתמשנו בגרסא 3.12.1).
2. יש להתקין את החבילות הרלוונטיות:

```
$ py -m pip install openpyxl pandas
```

3. יש לספק את קובץ הקלט - $data.xlsx$ באותה תיקיה של הקובץ main.py.
4. זמן הריצה עבור אוסף נקודות מוגדר לדקה, ניתן לשנות ע"י שינוי המשתנה MAX_RUNTIME_MINUTES.
5. יש להריץ את הקובץ main.py, ניתן לעקוב אחר התקדמות האלגוריתם בעזרת ההדפסות.
6. בסיום ריצת התוכנית, יתקבל קובץ $output.xlsx$, בו יהיו התוצאות. **כדי שהתוכנית תוכל לרשום את הפלט לקובץ, יש לשים לב שהוא אינו פתוח ברקע, אחרת הכתיבה נכשלת.**