פתרון בעיות באלגוריתמים 2020/2021

שיעור 1 – 19.10.2020

בעיית ה-Range Minimum Query (RMQ):

בהינתן מערך A, מעוניינים לענות על שאילתה מהצורה "מהו המינימום בקטע $A[i, \dots, j]$?" עבור $i \leq j$.

דוגמה:

2	1	5	6

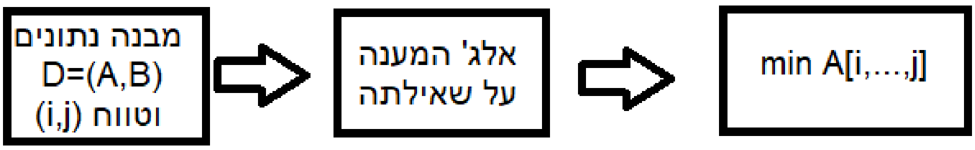
A=, שואלים אותנו "מה המינימום בטווח $[1,3]$?". התשובה: 1.

מדובר בבעיה שמכילה שאילתות, לכן נרצה לבצע עיבוד מקדים כלשהו למערך, כלומר, לחלץ ממנו מידע מסוים ואז להשתמש במידע זה כדי לענות על כל שאילתה שנקבל בצורה יעילה.
שאלות כאלו, שכוללות עיבוד מוקדם ואז שאילתות שמתבססות עליו, כוללות בד"כ שני אלגוריתמים:

1. אלגוריתם עיבוד מוקדם, אותו נריץ פעם אחת:

B הוא מבנה הנתונים שנקבל לאחר העיבוד המוקדם, בעזרתו יהיה נוח לענות על שאילתות.

2. אלגוריתם מענה על שאילתה, אותו נריץ הרבה פעמים:



נציג מספר פתרונות לבעיה, כאשר נשתמש במדדים הבאים על מנת לנתח את הסיבוכיות שלהם :

- T_p - (t עבור time, p עבור pre-processing)זמן הריצה של אלגוריתם **העיבוד המוקדם**.
- T_q – (t עבור time, q עבור query) זמן הריצה של אלגוריתם **המענה על השאילתה**.
- S – (S עבור space) כמות **המקום שמבנה הנתונים D** תופס.

נתעד את ניתוחי הפתרונות שלנו בטבלה הבאה:

שם הפתרון	T_p	T_q	S

RMQ1 – ללא עיבוד מוקדם:

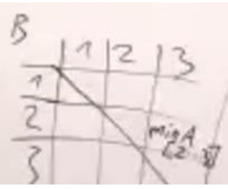
- **עיבוד מוקדם** – ללא, בעלות $O(1)$.
- **מענה על שאילתה** – מציאת מינימום בקטע הנתון בעלות לינארית, כלומר, $O(n)$.

כרגע, טבלת ניתוח הפתרונות שלנו נראית כך:

שם הפתרון	T_p	T_q	S
RMQ1	$O(1)$	$O(n)$	$O(n)$

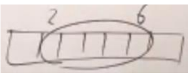
(המקום הוא $O(n)$ משום שאנחנו מתייחסים לפתרון כאילו כן יש לו שלב של עיבוד מקדים שרק מקבל ומחזיר את A מבלי לעשות לו כלום.

לכן, מבנה הנתונים הוא בגודל $O(n)$).



RMQ2 -הרבה עיבוד מוקדם:

- **עיבוד מוקדם** – נגדיר טבלה $B[i,j] = min\ A[i,\dots,j]$ בגודל $n \times n$ נמלא את הטבלה בעזרת תכנון דינאמי. נביט בדוגמה הבאה כדי להבין את הרעיון מאחורי התכנון הדינאמי:

נניח שנתון לנו מערך ואנחנו רוצים למצוא את האיבר המינימלי בקטע $[2,6]$ .

אנחנו מניחים שכבר חישבנו את השאילתה עבור קטעים קצרים יותר, לכן יש שתי תשובות אפשריות לשאילתה:

1. האיבר המינימלי בקטע $A[2,5]$.
2. $A[6]$.

משווים בין שתי האפשרויות ומחזירים כתשובה את המינימלית מבניהן.

נמלא את הטבלה באופן הבא:

$$B[i,j] = \begin{cases} A[j] & i = j \\ min\{B[i,j-1], A[j]\} & j > i \end{cases}$$

- **מענה על שאילתה** – החזר את $B[i,j]$.

- T_p - $\boldsymbol{O(n^2)}$ כי ממלאים טבלה בגודל n^2, כאשר את מקרי הבסיס (האלכסון) ממלאים ב-$O(1)$ ואת הצעד גם (באמצעות השוואת שני ערכים).
- T_q – $\boldsymbol{O(1)}$.
- S – $\boldsymbol{O(n^2)}$, כי המערך A תופס n ו-B תופס n^2.

שם הפתרון	T_p	T_q	S
RMQ1	$O(1)$	$O(n)$	$O(n)$
RMQ2	$O(n^2)$	$O(1)$	$O(n^2)$

RMQ3 – חלוקה לבלוקים בגודל $\sqrt{n}$ ("פשרה" בין שני הפתרונות הקודמים)

עד עכשיו ראינו שתי גישות קיצון – הראשונה משקיעה את כל העבודה בעיבוד מקדים ושום עבודה במענה על שאילתה, והשנייה לא עושה שום עיבוד מקדים ועובדת רק בעת קבלת שאילתה.
ב-RMQ3, נשקיע מאמץ בכל עיבוד שאילתה, אך נטיל חלק מהעומס על עיבוד מקדים לא כבד מדי.

- **עיבוד מוקדם** – חלוקת המערך ל-$\sqrt{n}$ בלוקים בגודל $\sqrt{n}$ כל אחד וחישוב מינימום לכל בלוק.
  את ערכי המינימום של כל בלוק נחזיק במערך חדש B.
  (נשים לב שבחרנו $\sqrt{n}$ כדי שלא יהיו לנו הרבה בלוקים ושכל אחד מהם לא יהיה גדול מדי).

- **מענה על שאילתה** –
  1. נביע את המקטע כאיחוד זר של בלוקים ונמצא את המינימום עבורם בעזרת המערך B.
  2. נמצא מינימום ב"זנבות" (הקצוות של הטווח שאולי לא מהווים בלוק שלם – ראו דוגמה).
  3. נחזיר את האיבר המינימלי שהתקבל עבור קבוצת ערכי המינימום של הבלוקים והזנבות.

נביט בדוגמה:

נניח שקיבלנו שאילתה על הטווח המסומן בצהוב. המערך בגודל 16, לכן נחלקו ל-4 בלוקים בגודל 4.

2	3	5	1	2	5	7	2	9	7	8	2	1	3	5	7

- קודם כל, נשים לב שהבלוקים שמוכלים במלואם הם השני והשלישי. בשלב העיבוד המוקדם כבר חישבנו כי המינימום של כל אחד מהם הוא 2.

B =

1	2	2	1

- עכשיו עלינו להשוות את 2 ל-2 ו-2 למינימום ב"זנבות" – החצי השני של בלוק 1 והחצי הראשון של בלוק 4. נשים לב שאנחנו לא יכולים להסתמך על המידע מהעיבוד המוקדם בנוגע למינימום של בלוקים 1 ו-4 כולם, כי יש איברים שלא רלוונטיים אולי לשאילתה שלנו.
- נצטרך לחשב את המינימום על הזנבות בעצמנו - זה לא נורא כי הם רק שניים והם קצרים.

- T_p - $\boldsymbol{O(n)}$, לחשב מינימום של כל בלוק זה בערך כמו לחשב מינימום של כל המערך.
- T_q – $\boldsymbol{O(\sqrt{n})}$:
  1. מציאת מינימום במערך B עולה $O(\sqrt{n})$.

2. מציאת המינימום בזנבות עולה גם $O(\sqrt{n})$ – נשים לב שכל זנב הוא בגודל של לכל היותר $\sqrt{n}$, אחרת הוא מכיל בתוכו בלוק שלם והוא לא זנב. אז יש לנו שני זנבות לכל היותר באורך $\sqrt{n}$, לכן מציאת מינימום בהם עולה $O(\sqrt{n})$.

- S – $\boldsymbol{O(n)}$, כי המערך המקורי תופס n ו-B תופס $\sqrt{n}$**.**

<u>סיכום ביניים</u> – הטבלה שלנו עד כה נראית כך:

S	T_q	T_p	שם הפתרון
$O(n)$	$O(n)$	$O(1)$	**RMQ1**
$O(n^2)$	$O(1)$	$O(n^2)$	**RMQ2**
$O(n)$	$O(\sqrt{n})$	$O(n)$	**RMQ3**

שיעור 2 – 26.10.2020

נגדיר סימון חדש שמייצג את טבלת ניתוח הסיבוכיות איתה עבדנו עד עכשיו: $(\mathrm{T_p}, \mathrm{T_q}, \mathrm{S})$. למשל, עבור RMQ1 נכתוב $(O(1), O(n), O(n))$.

RMQ4 – חלוקה לקטעים באורכי חזקות של 2:

- **עיבוד מוקדם –**

1. נגדיר טבלה B עם

$$\boxed{\mathbf{B[i,k] = min\{A[i, \dots, i + 2^{k} - 1]\}}}$$

אפשר לחשוב על הטבלה כך:

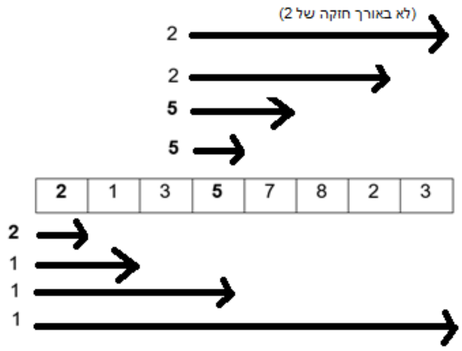
$\mathrm{B[start\ index, log(segment's\ length)]}$

כלומר, בתא $\mathrm{B[i,k]}$ יהיה המינימום בקטע בגודל 2^k המתחיל ב-$\mathrm{A[i]}$.

גודל B הוא $O(n \cdot \log(n))$, שכן עבור n ערכים שומרים $\log(n)$ ערכים.

*(אם עבור I ו-k מסוימים המרחק בין $\mathrm{A[i]}$ לסוף A קטן מ-2^k, נשמור ב-$\mathrm{B[i,k]}$ את המינימום בקטע הארוך ביותר האפשרי. זה אומר שיכול להיות שבשורה מסוימת, החל מעמודה מסוימת ימינה והלאה נראה את אותו המספר.)

<u>דוגמה:</u> עבור המערך הבא, נביט בשורות של B עבור $i = 0$ ו-$i = 3$, כלומר עבור הערכים 2 ו-5 במערך A. כדי להבין על אילו קטעים לוקחים מינימום, נצייר חצים לאורכם ומשמאל לחצים את המינימום של כל קטע.



אפשר לחשוב על הטבלה כך:

$\mathrm{B[start\ index, log(segment's\ length)]}$

זו הטבלה שנקבל:

i \ K	0	1	2	3
0	2	1	1	1
⋮	⋮	⋮	⋮	⋮
3	5	5	2	2
⋮	⋮	⋮	⋮	⋮

2. מילוי הטבלה – בעזרת תכנון דינאמי:

$$\boxed{\mathbf{B[i,k]} = \begin{cases} \mathbf{A[i]} & \mathbf{k = 0} \\ \mathbf{min\{B[i, k-1], B[i + 2^{k-1}, k-1]\}} & \mathbf{k > 0} \end{cases}}$$

אנחנו ממלאים את מקרי הבסיס ואת מקרי הצעד ב-$O(1)$, כלומר, אנחנו ממלאים טבלה בגודל $O(n\cdot log(n))$, כל תא ב-$O(1)$, לכן **מילוי הטבלה יעלה** $\boldsymbol{O(n\cdot log(n))}$.

3. חישוב טבלה עם ערכי $\lfloor log(t)\rfloor$ לכל t רלוונטי בעזרת תכנון דינאמי (ראה הגדרת t ושימושיו בחלק של מענה על שאילתה) באופן הבא: אנחנו יודעים לחשב בקלות את החזקות של 2, אז לפי החישוב הזה נמלא בטבלה של ערכי $\lfloor \boldsymbol{log(t)}\rfloor$ את הערכים שהם חזקות שלמות של 2 ואת ערכי הביניים נמלא לפי החזקה השלמה הקרובה משמאל.

t	1	2	3	4	5	6	7	8	…
$\lfloor \boldsymbol{log(t)}\rfloor$	0	1	1	2	2	2	2	3	…

- **מענה על שאילתה** – אנחנו יודעים לענות בצורה נוחה על שאילתות באורך חזקה של 2 (כי התשובות נמצאות לנו ב-B), איך נענה על שאילות באורך שאינו חזקה של 2?

<u>אינטואיציה</u>: יכול להיות שאורך הקטע שנקבל הוא לא חזקה של 2 ולכן אין לנו תשובה מוכנה בטבלה. במקרה זה, נרצה להחזיר את המינימום בין ערכי מינימום של אינטואיציה שני קטעים שכבר חישבנו: אחד מתחילת הקטע קדימה ואחד מסוף הקטע אחורה. חשוב לנו ש:

- התת-קטע מההתחלה לא יחרוג מסוף הקטע.
- התת-קטע מסוף הקטע לא יעבור את תחילת הקטע.
- לא נפספס איברים באמצע הקטע.

נסמן: $\boxed{\mathbf{t = j - i + 1}}$

החזר את

$$\boxed{\boldsymbol{min}\{\mathbf{B}[\mathbf{i}, \lfloor \mathbf{log(t)}\rfloor], \mathbf{B}[\mathbf{j} - \mathbf{2}^{\lfloor \mathbf{log(t)}\rfloor} + \mathbf{1}, \lfloor \boldsymbol{log(t)}\rfloor]\}}$$

*נשים לב שאת $\lfloor log(t)\rfloor$ אפשר לחשב ב-$O(1)$ בזכות שלב 3 בעיבוד המקדים.

<u>נוכיח את נכונות המענה על שאילתה:</u>

כאמור באינטואיציה, יש 3 דברים שחשובים לנו ונרצה להוכיח שמתקיימים לגבי שני התת-קטעים שאנחנו לוקחים ובפרט לגבי $log(t)$ (בחירת גודל זה קובעת איזה תת-קטעים ניקח):

1. אם מתקדמים $\mathbf{2^{\lfloor log(t)\rfloor} - 1}$ איברים מתחילת הקטע ($\mathrm{A[i]}$) לא חורגים מסופו ($\mathrm{A[j]}$).
2. אם "הולכים אחורה" $\mathbf{2^{\lfloor log(t)\rfloor} - 1}$ איברים מסוף המערך, לא עוברים את תחילתו.
3. לא מפספסים איברים בין סוף הקטע הראשון ותחילת הקטע השני.

נוכיח את הסעיפים לעיל:

<u>הבחנות:</u>

$(*)\ \boxed{2^{\lfloor \log(t) \rfloor} \le 2^{\log(t)} = t}$

$(**)\ \boxed{2^{\lfloor \log(t) \rfloor + 1} \ge t + 1} \Leftarrow 2^{\lfloor \log(t) \rfloor + 1} > 2^{\log(t)} = t$

1. נשתמש ב-(*) כדי להראות שהתת-קטע הראשון לא חורג מסוף הקטע, כלומר נראה שמתקיים $\mathbf{i + 2^{\lfloor \log(t) \rfloor} - 1 \le j}$:

$$\mathbf{i + 2^{\lfloor \log(t) \rfloor} - 1} \le \mathrm{i} + 2^{\log(\mathrm{t})} - 1 = \mathrm{i} + 2^{\log(\mathrm{j-i+1})} - 1 = \mathrm{i + j - i + 1 - 1} = \mathbf{j}$$

2. נשתמש ב-(**) כדי להראות שהתת-קטע השני לא חורג מתחילת הקטע, כלומר, נראה שמתקיים: $\boldsymbol{j - 2^{\lfloor \log(t) \rfloor} + 1 \ge i}$:

$$\boldsymbol{j - 2^{\lfloor \log(t) \rfloor} + 1} \ge j - 2^{\log(\mathrm{t})} + 1 = j - 2^{\log(\mathrm{j-i+1})} + 1 = j - j + i - 1 + 1 = \boldsymbol{i}$$

3. כעת, נראה שאנחנו לא מפספסים איברים בין שני התת-קטעים, כלומר, נרצה להראות שהקצה הימני של התת-קטע ההתחלתי גדול-שווה לקצה אינטואיציה השמאלי של התת-קטע השני. פורמלית, נרצה להראות כי $\boldsymbol{i + 2^{\lfloor \log(t) \rfloor} - 1 \ge j - 2^{\lfloor \log(t) \rfloor} + 1}$:

$$\boldsymbol{i + 2^{\lfloor \log(t) \rfloor} - 1 \ge j - 2^{\lfloor \log(t) \rfloor} + 1} \iff$$
$$\boldsymbol{2^{\lfloor \log(t) \rfloor} + 2^{\lfloor \log(t) \rfloor} \ge j - i + 1 + 1} \iff$$
$$\boldsymbol{2^{\lfloor \log(t) \rfloor + 1} \ge \underbrace{(j - i + 1)}_{t} + 1 = t + 1} \iff (**)$$

נשים לב שבקצה הימני של הגרירה קיבלנו את (**), לכן הצד השמאלי של הגרירה נכון.

לסיכום הנושא, טבלת הזמנים שלנו נראית כך:

שם הפתרון	T_p	T_q	S
RMQ1	$O(1)$	$O(n)$	$O(n)$
RMQ2	$O(n^2)$	$O(1)$	$O(n^2)$
RMQ3	$O(n)$	$O(\sqrt{n})$	$O(n)$
RMQ4	$O(n \cdot log(n))$	$O(1)$	$O(n \cdot log(n))$

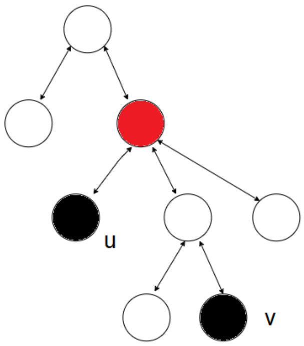
בעיית ה-LCA (אב קדמון משותף מינימלי - Lowest Common Ancestor)

הגדרות:

- **אב קדמון** u של קודקוד v בעץ, הוא קודקוד כלשהו במעלה העץ אשר v צאצא של u (בפרט, v הוא אב קדמון של עצמו).
- **אב קדום משותף** של u ו-v הוא אב קדמון גם של u וגם של v.
- **אב קדמון משותף מינימלי** של u ו-v הוא האב הקדמון המשותף של שני הקודקודים אשר נמצא הכי נמוך בעץ מתוך כל האבות הקדמונים המשותפים להם.

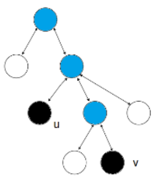
הגדרת הבעיה: בהינתן עץ מושרש (מדובר בעץ כללי – לא חיפוש ולא בינארי!), מעוניינים לענות על שאילתות מהצורה "מהו הקודקוד הנמוך ביותר ש-u ו-v צאצאיו?", בהינתן u ו-v בשאילתה.

(לאורך כל ההתעסקות עם הבעיה נניח כי v נמוך מ-u.)



דוגמה: עבור העץ הנתון, ה-LCA של שני הקודקודים המסומנים בשחור הוא הקודקוד המסומן באדום:

LCA1 – $(\boldsymbol{O(n)}, \boldsymbol{O(n)}, \boldsymbol{O(n)})$:



רעיון הפתרון:

1. התקדם מ-v עד לשורש וסמן קודקודים בדרך (למשל בכחול, כמו בתמונה).
2. בצע זאת עבור u.
3. החזר את הקודקוד הראשון אותו סימנו בשלב 2 וכבר היה מסומן בשלב 1.

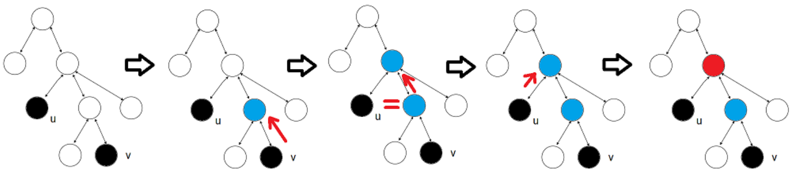
ניתוח סיבוכיות האלגוריתם:

- $\mathrm{T_p}$ - $\boldsymbol{O(n)}$, נקצה מערך בגודל $O(n)$ עבור תיעוד הקודקודים שפגשנו בדרך לשורש מ-v.
- $\mathrm{T_q}$ – $\boldsymbol{O(n)}$: מטפסים מ-v עד לשורש ומ-u עד לשורש, זה עולה $O(h)$ פעמיים. גובה העץ הוא לכל היותר מספר הקודקודים בעץ, ולכן $O(n)$ פעמיים.
- S – $\boldsymbol{O(n)}$, העץ עצמו + גודל המערך בו נתעד את הקודקודים שפגשנו הוא, כאמור, $O(n)$.

LCA2 - $(\boldsymbol{O(n)}, \boldsymbol{O(n)}, \boldsymbol{O(n)})$:

- **עיבוד מוקדם**: חשב לכל קודקוד את רמתו בעץ. זה קורה בעזרת DFS: מתחילים מהשורש (יודעים שרמתו היא 0), מתקדמים לבנים שלו – רמתם היא רמת השורש 1+, מכל בן מתקדמים לבנים שלו עם DFS וכו'. אפשר לבצע גם עם BFS.
- **מענה על שאילתה**: התקדם מהנמוך יותר עד להשוואת רמות עם הגבוה יותר ולאחר מכן טפס במקביל משניהם עד להגעה לקודקוד משותף.

דוגמה:



- T_p - $\boldsymbol{O(n)}$, נבצע DFS בעלות $O(n)$.
- T_q – $\boldsymbol{O(n)}$, מתקדמים מהנמוך יותר עד להשוואת רמות – $O(h)$, לכל היותר $O(n)$. לאחר מכן מטפסים משני קודקודים במקביל – גם לכל היותר $O(n)$.
- S – $\boldsymbol{O(n)}$, העץ עצמו + גודל המערך בו נתעד את הקודקודים שפגשנו הוא, כאמור, $O(n)$.

LCA3 – חלוקה לקבוצות רמות בגודל $\sqrt{h}$ כל אחת - $(\boldsymbol{O(n)}, \boldsymbol{O(\sqrt{n})}, \boldsymbol{O(n)})$:

נשאב השראה מ-RMQ3 ונרצה לחלק את העץ לקבוצות איכשהו. אולי אפשר היה לחשוב שכדאי לחלק את כל הקודקודים לקבוצות בגודל $\sqrt{n}$, אך משום שמדובר על אב קדמון מינימלי ולכן על רמות בעץ, יותר מתאים שנתמקד בגובה העץ.

לכן, נחלק את גובה העץ לבלוקים המכילים $\sqrt{h}$ רמות כל אחד, כך שכל בלוק מכיל אמנם מספר שונה של קודקודים אך מספר זהה של רמות.

לכן ננסה לעשות כמה שיותר קפיצות גדולות, שמדלגות על בלוקים שלמים של רמות, וכמה שפחות קפיצות קטנות מקודקוד לאביו.

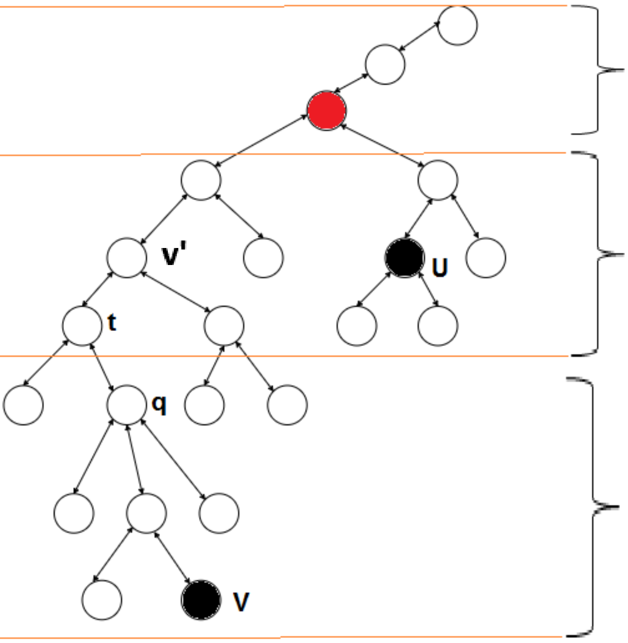
- **עיבוד מוקדם**:
  - כמו ב-LCA2, חשב לכל קודקוד את רמתו.
  - נחלק את גובה העץ לבלוקים המכילים $\sqrt{h}$ רמות כל אחד.
  - לכל קודקוד, חשב את האב הקדמון הנמוך ביותר מקבוצת הרמות שמעל זו שלו עצמו. את התוצאה שמור במערך $GP[v]$.
  - נמלא את GP בעזרת תכנון דינאמי:

$\mathbf{GP[v]}\begin{cases}\mathbf{v.parent} \\ \mathbf{GP[v.parent]}\end{cases}$	אם v הגבוה ביותר ברמתו אחרת

- **מענה על שאילתה**:
  - בצע $O(\sqrt{h})$ קפיצות גדולות, עד להגעה לקודקוד מקבוצת הרמות של u.
  - בצע $O(\sqrt{h})$ קפיצות קטנות להשוואת רמות u ו-v (כלומר, הקודקוד הנוכחי במסלול מ-v).
  - בצע $O(\sqrt{h})$ קפיצות גדולות מ-u ו-v במקביל עד להגעה לקודקוד משותף (בשלב הזה הגענו לקודקוד שהוא הכי נמוך בקבוצתו וכן אב קדמון משותף, אל לאו דווקא מינימלי).
  - בצע $O(\sqrt{h})$ קפיצות קטנות מזוג הקודקודים הקטנים שהתקבלו בשלב הקודם (כלומר, לפני ההגעה לאב הקדמון המשותף – לא ביצענו את הקפיצה האחרונה בפועל) עד ל הגעה ל-LCA.

<u>דוגמה:</u>

עבור הגרף משמאל, עבור הקודקודים u,q ו-t הטבלה GP תראה כך:



קודקוד	אב קדמון נמוך ביותר ברמה מעל
⋮	⋮
q	t
⋮	⋮
v	t
⋮	⋮

מענה על שאילתה:

- נעלה מ-v עד ל-t.
- נעלה מ-t עד ל-'v.
- נזהה שאם נקפוץ במקביל מ-u ומ-'v קפיצה גדולה אחת נגיע לקודקוד האדום, שהוא CA.
- נקפוץ מ-u ומ-'v קפיצות קטנות עד שנגיע לקודקוד האדום המשותף, אותו נחזיר כ-LCA.

- T_p - $\boldsymbol{O(n)}$, נבצע DFS בעלות $O(n)$ ונמלא טבלה בגודל $O(n)$, כל תא ב-$O(1)$.
- T_q – $\boldsymbol{O(\sqrt{n})}$, כמתואר לעיל, כל שלב בתהליך עולה $\boldsymbol{O(\sqrt{h})} \le \boldsymbol{O(\sqrt{n})}$.
- S – $\boldsymbol{O(n)}$, העץ עצמו + גודל GP, שגודלה $O(n)$.

LCA4 – חישוב אב קדמון במרחק 2^k לכל קודקוד ולכל k – :$\left(\boldsymbol{O}(\boldsymbol{n}\cdot\boldsymbol{log}(\boldsymbol{n})),\boldsymbol{O}(\boldsymbol{log}(\boldsymbol{n})),\boldsymbol{O}(\boldsymbol{n}\cdot\boldsymbol{log}(\boldsymbol{n}))\right)$

בדומה ל-RMQ4 נרצה, במקום לחלק את העץ לבלוקים, לחשב עבור כל קודקוד משהו במרחק 2^k ממנו (לכל k שהוא לא גדול מדי). די טבעי לחשב לכל קודקוד את האב הקדמון במרחק 2^k ממנו.

- **עיבוד מוקדם:**
  - חשב לכל קודקוד את רמתו.
  - חשב טבלה B כך ש-$\mathrm{B[v,k]}$ יכיל את האב הקדמון של v במרחק 2^k ממנו.

    נמלא את הטבלה באמצעות תכנון דינאמי באופן הבא:

$$\boxed{\mathrm{B[v,k]}\begin{cases}\mathrm{v.parent} & \mathrm{k}=0\\ \mathrm{B[B[v,k-1]},k-1] & else\end{cases}}$$

כלומר, עבור $\mathrm{k}\neq 0$ נחלק את הקפיצה לאב הקדמון במרחק 2^k מ-v לשתי קפיצות בגודל 2^{k-1}: קודם קופצים 2^{k-1} רמות מ-v באמצעות הסתכלות על $\mathrm{B[v,k-1]}$ ואז מהקודקוד אליו הגענו קופצים שוב 2^{k-1} באמצעות הסתכלות על $\mathrm{B[B[v,k-1]},k-1]$.

- **מענה על שאילתא:** הרעיון הוא לבצע חיפוש בינארי על העץ
  - לכל k מ-$(\log(h)-1)$ עד 0 (נשים לב כי $(\log(h)-1)$ מקיים $2^{\log(h)-1}=\frac{h}{2}$):
    - אם $\mathrm{B[v,k]}$ אינו גבוה מ-u עדכן $\mathrm{v}\leftarrow\mathrm{B[v,k]}$.

    שלב זה יביא אותנו לאב הקדמון של v שהוא באותה רמה כמו u.
  - לכל k מ-$(\log(h)-1)$ עד 0:
    - אם $\mathrm{B[v,k]}\neq\mathrm{B[u,k]}$ עדכן $\mathrm{v}\leftarrow\mathrm{B[v,k]}$ ועדכן $\mathrm{u}\leftarrow\mathrm{B[u,k]}$.
  - החזר את v.parent (ששווה ל-u.parent) בתור ה-LCA.

- $\mathrm{T_p}$ - $\boldsymbol{O}(\boldsymbol{n}\cdot\boldsymbol{log}(\boldsymbol{n}))$, יש n שורות ב-B ובכל שורה יש $\log(h)$ תאים. כלומר, גודל B הוא $O(n\cdot\log(h))=O(n\cdot\log(n))$ וממלאים כל תא ב-$O(1)$, לכן זמן הריצה $O(n\cdot\log(n))$.
- $\mathrm{T_q}$ – $\boldsymbol{O}(\boldsymbol{log}(\boldsymbol{n}))$, יש לנו $O(\log(n))$ איטרציות, כל איטרציה עולה $O(1)$.
- S – $\boldsymbol{O}(\boldsymbol{n}\cdot\boldsymbol{log}(\boldsymbol{n}))$, כפי שמתואר בניתוח $\mathrm{T_p}$, גודל B הוא $O(n\cdot log(n))$.

שיעור 3 – 2.11.2020

אלגוריתמים שזמן ריצתם כולל שורש של גודל הקלט

1. פירוק שורש (לאו דווקא ריבועי)

בטכניקה זו, מחלקים את הקלט (או פונקציה שלו) לבלוקים בגודל שורש הקלט כשלב מוקדם[1] ובהמשך עונים על שאילתות המשתמשות בחלוקה זו ברמת המאקרו[2] והמיקרו[3].
אפשר להכליל את הרעיון לעוד מבנים מעבר לעצים ומערכים – כמו למשל רשימות.
כמו כן, הרעיון תקף לא רק לשורש ריבועי, אלא גם לשורשים מסדרים אחרים – זה לפעמים שימושי: למשל לפעמים נתקל בבעיה דו-ממדית ובה נצטרך שורש ריבועי, בבעיה תלת-ממדית אולי נצטרך שורש מסדר שלוש וכו'. הפרטים הספציפיים של השיטה הם תלויי-בעיה ובד"כ נפתרים ע"י בעיית אופטימיזציה מתאימה כלשהי.
בנוסף, גרסה מוכללת של רעיון פרוק השורש יכולה לחלק את הקלט לבלוקים שאינם זרים, כלומר, יש ביניהם חפיפה.

2. שילוב אלגוריתמים

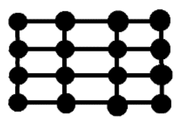
טכניקה זו שימושית במקרים בהם לבעיה נתונה ידוע אלגוריתם שטוב עבור חלק מהקלטים, אך לא עבור היתר. מוצאים אלגוריתם נוסף שטוב עבור היתר, משלבים בין השניים לקבלת אלגוריתם שטוב משניהם.

<u>נראה בעיה ספציפית:</u>

בהינתן טבלה עם אותיות, מעוניינים למצוא את זוג התאים הקרובים ביותר שמכילים את אותה האות. נציין שהמטריקה היא מרחק מנהטן – מודדים רק התקדמות אנכית או אופקית ולא באלכסון ($|x_1 - x_2| + |y_1 - y_2|$).

A	F	B	A
C	E	G	E
B	D	A	F
A	C	B	D

ניתן לחשוב על הטבלה כגרף בצורת סריג כאשר אפשר ללכת רק לאורך הצלעות שלו.



(עבור הדוגמה משמאל, התשובה היא שני תאים במרחק 2 המכילים E).

---

[1] באלגוריתמים שראינו קראנו לזה "עיבוד מוקדם" אך זה יכול להיות סתם שלב מוקדם של האלגוריתם, למשל שלב מוקדם במילוי טבלה.
[2] הכוונה לאוסף בלוקים - למשל שימוש במערך ערכי המינימום של הבלוקים ב-RMQ3.
[3] כי לפעמים החלק שנדבר עליו לא "נתפס" ע"י העיבוד המוקדם – כמו הזנבות ב-RMQ3.

<u>אלגוריתם 1:</u>

נמיין[4] את הקלט לפי האותיות ולכל זוג תאים עם אותה האות נמצא את המרחק ונחזיר את המינימום על פני כל הדברים שמצאנו.

- המקרה הכי קשה עבור האלגוריתם הזה הוא המקרה בו בכל התאים יש את אותה האות. במקרה זה, נשווה כל תא לכל שאר התאים ולכן זמן הריצה יהיה $\boldsymbol{O(n^2)}$.
- לעומת זאת, אם בכל תא יש אות אחרת אז לא צריך להשוות אף תא לאף תא אחר ואז זמן הריצה הוא $O(n \cdot log(n))$ – בגלל שהמיון עולה $\boldsymbol{O(n \cdot log(n))}$.

<u>אלגוריתם 2:</u>

- מיין את הטבלה.
- הרץ BFS במקביל מכל המופעים של אותה האות למציאת המינימום עבורה.
- החזר את המינימום של ערכי המינימום.

<u>זמן ריצה של אלגוריתם 1:</u>

נסמן ב-m את כמות סוגי האותיות ואת מספר המופעים של האות ה-i ב-k_i.

אז זמן הריצה מורכב ממיון והשוואת כל תא עם אות i לכל שאר התאים עם האות i:

$$\boldsymbol{O\left(n \cdot log(n) + \sum_{i=1}^{m} {k_i}^2\right)}$$

זמן ריצה זה עלול להיות $O(n^2)$ כאשר בכל התאים יש את אותה האות.

<u>זמן הריצה של אלגוריתם 2:</u>

שוב נסמן ב-m את כמות סוגי האותיות.

העלות של BFS היא $O(|V| + |E|)$, משום שמדובר בגריד ודרגת כל קודקוד קבועה זה שווה $O(n)$. כמו כן, משום שאנחנו מריצים BFS במקביל אף קודקוד לא נסרק פעמיים ולכן זמן הריצה של כל ההרצות המקבילות יחד נשאר $O(|V| + |E|)$.

אז זמן הריצה בנוי ממיון ומזמן הרצת BFS עבור כל סוג של אות (אולי יש כמה הרצות במקביל לכל אות, אבל זה לא שמייקר את הזמן אסימפטוטית). נקבל:

$$\boldsymbol{O(n \cdot log(n) + m \cdot n)}$$

[4] ממיינים את המטריצה למערך חד-ממדי בו כל תא מכיל קואורדינטות של תא במטריצה + את האות שבו. התאים ממוינים לפי האות.

<u>שילוב האלגוריתמים:</u>

- מיין את הטבלה למערך.
- הגדר קבוע k.
- לכל אות c_i עם מספר מופעים k_i המקיים $k > k_i$, הרץ את אלגוריתם 1.
  (כי אמרנו שהוא טוב עבור אותיות שיש להן מעט מופעים).
- עבור יתר סוגי האותיות, נריץ את אלגוריתם 2.

<u>זמן ריצת האלגוריתם המשולב:</u>

נסמן ב-m_1 את כמות סוגי האותיות שיש מהן מעט מופעים (פחות מ-k) וב-m_2 את כמות סוגי האותיות שיש מהן הרבה מופעים (k או יותר).

- עבור מעט מופעים:

$$\boldsymbol{O\left(n\cdot log(n)+\sum_{i=1}^{m_1} {k_i}^2\right) \le O\left(n\cdot log(n)+k\sum_{i=1}^{m_1} k_i\right) \le O(n\cdot log(n)+k\cdot n)}$$

- עבור היתר:
  נשים לב שיש לפחות $\frac{n}{k}$ סוגי אותיות עם לפחות k מופעים, אז:

$$\boldsymbol{O(n\cdot log(n)+m_2\cdot n) \le O\left(n\cdot log(n)+\frac{n}{k}\cdot n\right) = O\left(n\cdot log(n)+\frac{n^2}{k}\right)}$$

- זמן הריצה הכולל, שהוא סכום של שני הזמנים לעיל:

$$\boldsymbol{O\left(n\cdot log(n)+k\cdot n+\frac{n^2}{k}\right)}$$

נראה שתי דרכים למצוא את ה-k עבורו זמן הריצה יהיה מינימלי:

i. <u>מציאת המינימום של פונקציית זמן הריצה</u> (בעיקרון צריך גם לבצע נגזרת שנייה ולהוכיח שמדובר בנק' מינימום - לא נראה זאת):

$$\frac{\boldsymbol{d\left(n\cdot log(n)+k\cdot n+\frac{n^2}{k}\right)}}{\boldsymbol{dk}} = n-\boldsymbol{\frac{n^2}{k^2}} = \boldsymbol{0} \Leftrightarrow \boldsymbol{n} = \boldsymbol{\frac{n^2}{k^2}} \Leftrightarrow \boxed{\boldsymbol{k=\sqrt{n}}}$$

נציב את ה-k שקיבלנו ונראה כי זמן הריצה המינימלי הוא:

$$\boldsymbol{O\left(n\cdot log(n)+\sqrt{n}\cdot n+\frac{n^2}{\sqrt{n}}\right) = O\left(n\cdot log(n)+\sqrt{n}\cdot n+\sqrt{n}\cdot n\right) = \boxed{O(\sqrt{n}\cdot n)}}$$

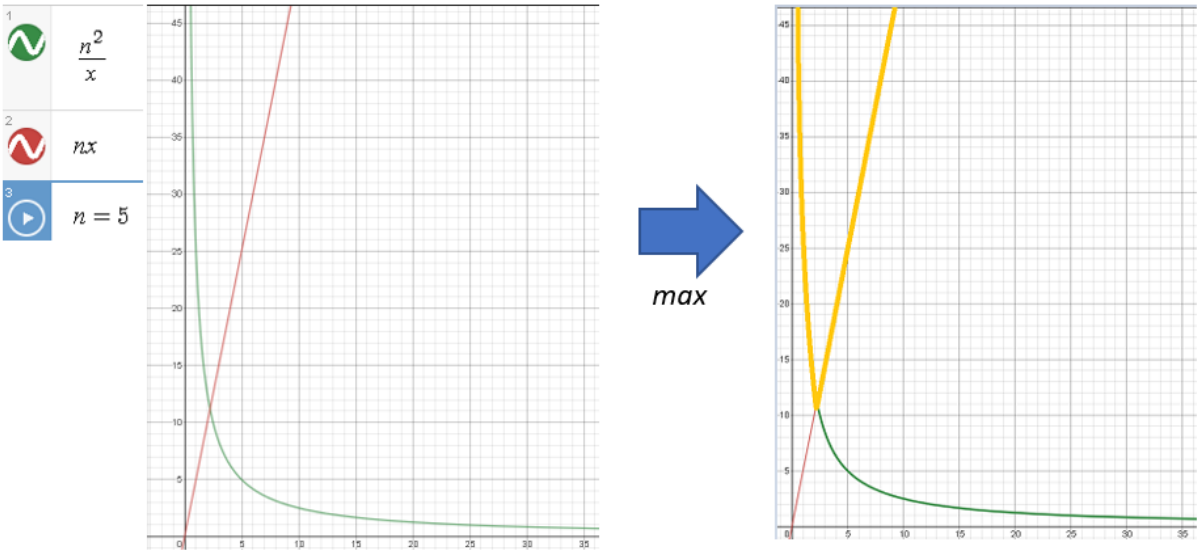
וזה זמן יותר טוב מ-$O(n^2)$, שזה זמן הריצה של שני האלגוריתמים הקודמים.

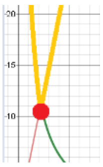
ii. <u>דרך שהיא יותר אינטואיציה גיאומטרית יפה (לא תמיד תסתדר יפה עם פו' אחרות):</u>

נעזר בעבודה "O גדול של סכום שווה ל-O גדול של מקסימום" ונכתוב:

$$\boldsymbol{O\left(n \cdot log(n) + k \cdot n + \frac{n^2}{k}\right) = O\left(max\left\{k \cdot n, \frac{n^2}{k}\right\}\right)}$$

אם נצייר את שתי הפונקציות האלו נקבל גרפים שצורתם בערך כמו הגרף למטה. פונקציית המקסימום של שתי הפונקציות האלו הוא החלק העליון המודגש בגרף הימני





אנחנו מחפשים את ערך ה-k שיביא למינימום את פונקציית המקסימום, כלומר את נקודת המינימום (הברורה ויזואלית) של הגרף הצהוב לעיל.

ערך זה הוא בדיוק נקודת המפגש בין שתי הפונקציות, לכן נרצה לחשב:

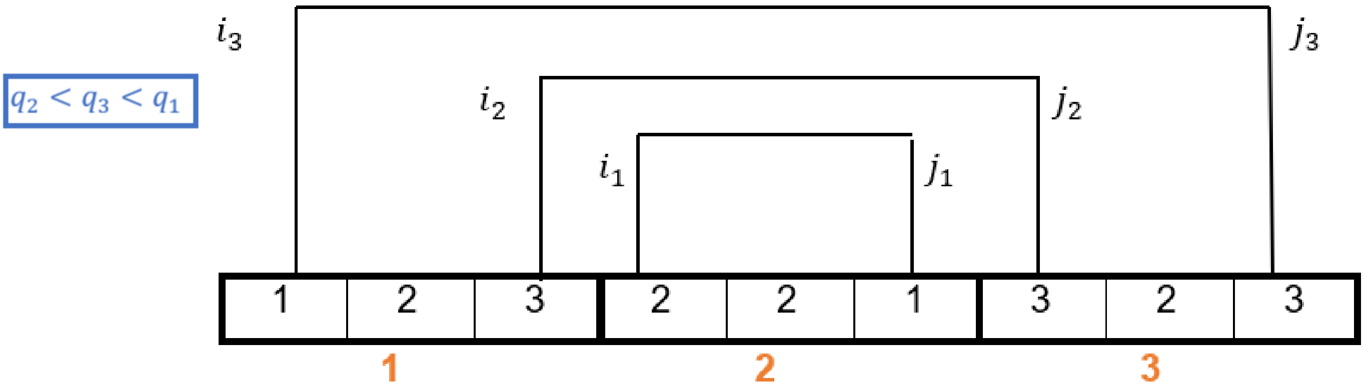
$$\boldsymbol{\frac{n^2}{k} = k \cdot n \Leftrightarrow n = k^2 \Leftrightarrow \boxed{k = \sqrt{n}}}$$

ואכן, עבור $n = 5$, למשל, מתקיים $k = \sqrt{5} = 2.36$:

3. <u>אלגוריתם Mo</u>

בטכניקה זו משתמשים בבעיות אופליין עם שאילתות טווח (כשהשאילתות ידועות מראש).

לצורך הדגמה נסתכל על המערך הבא (על אף שלא בהכרח מדובר במערך):



לצורך העניין אפשר לחשוב על שאילתות מבקשות למצוא מינימום או סכום בטווח.

בעיקרון, אנחנו רוצים לחלק את **השאילתות** לבלוקים, ולא את איברי המערך.

<u>סכמת האלגוריתם:</u>

i. **נמיין את השאילתות** לפי הקריטריון הבא - לכל זוג שאילתות $q_1 = (i_1, j_1),\ q_2 = (i_2, j_2)$:

א. אם[5] $\left\lfloor \frac{i_2}{\sqrt{n}} \right\rfloor < \left\lfloor \frac{i_1}{\sqrt{n}} \right\rfloor$ (הקצה השמאלי של q_1 בבלוק מוקדם מזה של q_2) אז q_1 לפני q_2.

(בדוגמה זה הפוך – i_2 בבלוק מוקדם מ-i_1, לכן q_2 לפני q_1).

ב. אם $\left\lfloor \frac{i_2}{\sqrt{n}} \right\rfloor = \left\lfloor \frac{i_1}{\sqrt{n}} \right\rfloor$ ואם $j_1 < j_2$ אז q_1 לפני q_2.

(במקרה שלנו שובר השוויון הזה מכתיב $q_2 < q_3$).

ii. **נמפה את השאילתות למקומן במערך הממוין** – אנחנו צריכים את המיפוי הזה כדי להחזיר למשתמש תשובות על השאילתות לפי הסדר בהן הוא נתן לנו אותן. כלומר, נענה על השאילתות לפי סדר המיון שלנו (תכף נראה למה זה נוח לנו), אבל את התשובות נרצה להחזיר לפי הסדר המקורי של השאילתות. למשל: נניח ש-q_1 היא רביעית במיון, אז אנחנו לא רוצים להחזיר את התשובה שלה כתשובה הרביעית, אלא כתשובה הראשונה. אז, כדי להחזיר את התשובה הראשונה (ל-q_1) ניגש לתא הרביעי במערך הממוין ונחזיר את התשובה לשאילתה שנמצאת שם.

---

[5] אם נחלק את המערך לבלוקים בגודל $\sqrt{n}$, אז $\left\lfloor \frac{i_1}{\sqrt{n}} \right\rfloor$ זה מספר הבלוק בו הקצה השמאלי של Q_1 נמצא.

iii. נחשב את תוצאות השאילתות לפי הסדר הממוין, כך שנשתמש בתוצאות משאילתה נתונה לטובת העוקבת לה, לרוב תוך שימוש במבנה נתונים נפרד.
למשל: יש לנו שתי שאילתות שמבקשות את הסכום בטווחים $[1,3]$ ו-$[1,4]$. נענה קודם על זו של $[1,3]$ ופשוט נוסיף לתוצאה שלה את האיבר במקום 4 כדי לענות על השנייה.
ההתקדמות על השאילתות היא בעזרת שני מצביעים: R ו-L.

דוגמה:

ספירת מס' האיברים השונים בטווח.

עבור המערך לעיל והשאילתות q_1, q_2, q_3 המתוארות באיור התשובה צריכה להיות: (2,3,3).

נריץ את האלגוריתם על הדוגמה:

- נניח כי האיברים בטווח $1, \dots, n$ (לא הנחה הכרחית, אך נוח להציג את הדברים כך).
- עם תחילת הריצה נאתחל ל-0 מונה של מספר האיברים השונים, אותו נחזיר בסופו של דבר כתשובה לכל שאילתה.
- נגדיר טבלה ששומרת לכל ערך את מספר ההופעות שלו עד כה. במעבר משאילתה אחת לעוקבת, אם R זז ימינה או L זז שמאלה – נגדיל ערך בטבלה. אחרת (R זז שמאלה או L זז ימינה), נקטין ערך בטבלה.
- אם ערך איבר בטבלה גדל מאפס לחיובי, נגדיל את המונה בתשובה של השאילתה הקודמת באחד (למשל, בדוגמה הספציפית שלנו התשובה ל-q_1 היא (1,2,0) , כאשר מזיזים את R ימינה הערך של 3 בטבלה עולה מ-0 ל-1 ולכן נגדיל את המונה של 3 בתשובה של q_1 כך שהתשובה של q_2, נכון לשלב זה של עיבודה, תהיה (1,2,1)). אם זו השאילתה הראשונה, פשוט נשנה את המונה שאתחלנו בשלב השני.
- אם חיובי הפך לאפס, נקטין את המונה מהשאילתה הקודמת באחד.

<u>ניתוח זמן ריצה</u>:

נסמן ב-m את כמות השאילתות וב-n את גודל המערך.

1. **מיון בעלות** $O(n+m)$ (כאשר ממיינים בעזרת מיון מנייה).
2. **סריקת המערך הממוין** $O(m)$ (אם רוצים לדעת עבור שאילתה מסוימת מה האינדקס שלה במערך הממוין, צריך לסרוק את המערך הממוין פעם אחת).
3. **עלות הזזת R -** $O(n \cdot \sqrt{n})$**:**

   - כאשר L בבלוק נתון, R זז ימינה $O(n)$ צעדים (מיינו במקרים האלה לפי מיקום הקצה הימני של הקטע, לכן במעבר על המערך הממוין של השאילתות R יזוז רק ימינה).

- במעבר של L מבלוק אחד לעוקב, R זז $\boldsymbol{O(n)}$ צעדים שמאלה (ברגע שסיימנו עם בלוק נתון, עוברים לבלוק הבא עם L ואז צריך לאפס את R ולאחר מכן להזיז אותו ימינה עד לקצה הימני של השאילתה הראשונה בבלוק החדש, ההליך הזה מטפל במצב בו יש שתי שאילתות q_1 ו-q_2 כך שהראשונה מתחילה בבלוק מוקדם מהשנייה אך מסתיימת לפניה).

אם כך, הזזת R עולה לנו $\boldsymbol{O(n \cdot \sqrt{n})}$ – עבור בלוק אחד R זז $O(n)$ צעדים (איפוס במעבר מבלוק אחד לעוקב + הזזת R ימינה במעבר משאילתה לשאילתה באותו בלוק) ויש לנו $\sqrt{n}$ בלוקים.

4. **עלות הזזת L** - $\mathbf{O(n + m \cdot \sqrt{n})}$:
    - במעבר בין בלוקים, L זז $O(n)$ צעדים ימינה (השאילתות ממוינות לפי הבלוקים בהם הקצה השמאלי שלהן נמצא, לכן במעבר על השאילתות הממוינות לא יקרה מצב בו מחזירים את L לבלוק שמאלי לזה בו הוא נמצא).
    - בבלוק נתון L זז $\mathrm{O}(\sqrt{\mathrm{n}})$ צעדים (כי גודל הבלוק הוא $\sqrt{n}$), אז את כל התזוזות של L בתוך בלוקים ניתן לחסום ע"י $\mathrm{O}(\mathrm{m} \cdot \sqrt{\mathrm{n}})$ (כי עבור כל שאילתה הוא זז לכל היותר בלוק שלם אחד והתזוזה מעבר לזה נכללת בתת-סעיף הקודם).

    בסך הכל, זמן הריצה של L שנקבל הוא: $\mathbf{O}(\boldsymbol{n} + \mathbf{m} \cdot \sqrt{\mathbf{n}})$.

<u>זמן ריצה כולל:</u>

הנחנו שתזוזה של R או L עולה לנו $O(1)$, אבל במקרה הכללי אפשר להניח שהיא עולה לנו t, לכן נכפיל את זמן הריצה שחישבנו עבור תזוזות הסמנים פי t ונקבל:

$$\boldsymbol{O}\left(\boldsymbol{n} + \boldsymbol{m} + \boldsymbol{m} + \boldsymbol{t}\left(\boldsymbol{n} \cdot \sqrt{\boldsymbol{n}} + \boldsymbol{n} + \boldsymbol{m}\sqrt{\boldsymbol{n}}\right)\right) = \boxed{\boldsymbol{O}\left(\boldsymbol{t} \cdot (\boldsymbol{n} + \boldsymbol{m}) \cdot \sqrt{\boldsymbol{n}}\right)}$$

אם מספר השאילתות הוא כגודל המערך קיבלנו אלגוריתם שעובד ב-$O(n\sqrt{n})$.

<u>דוגמה נוספת:</u> אפשר לחשוב על בעיה נוספת שאלגוריתם Mo יתאים לה – חישוב סכום על פני קטע. זוכרים סכום נתון, כל פעם ש-R זז ימינה או L זז שמאלה מגדילים את מונה הסכום בערך האיבר החדש שמצאנו, וכאשר R זז שמאלה או L זז ימינה מחסירים מהסכום את האיבר ש"איבדנו".

שיעור 4 - 9.11.2020

האם המיון שהצענו לאלגוריתם Mo הוא הכי טוב? (הערה)

התשובה היא **כן** – בהנחה שמספר השאילתות הוא גודל המערך.

אפשר להראות ב-worst case אי אפשר לעשות יותר טוב מהמיון הספציפי הזה.

הרעיון הוא לבנות דוגמה בה האלגוריתם שלנו משיג את התוצאה הכי טובה שאפשר.

אנחנו רוצים להראות דוגמה שזמן הריצה של כל אלגוריתם עליה יהיה $O\left(n \cdot \sqrt{n}\right)$, ומשום שזה זמן הריצה של האלגוריתם שלנו – ניצחנו.

<u>תזכורת:</u> כשדיברנו על אלגוריתם Mo מיינו את כל השאילתות – מיון ראשוני לפי מיקום הבלוק בו מתחיל טווח השאילתה ומיון שניוני לפי מיקום סוף הקטע במערך. כאמור, הרעיון הוא להשתמש בתשובה לשאילתה אחת כאיזשהו רכיב בתשובה לשאילתה אחרת ואנחנו רוצים ששאילתות קרובות תהיינה כמה שיותר דומות על מנת לחסוך חישובים. כמו כן, חילקנו את המערך לבלוקים בגודל $\sqrt{n}$.

נניח כי $m - \Theta(n)$, כלומר, מספר השאילתות הוא בערך גודל המערך, אז האלגוריתם שלנו עובד ב-$O\left((m+n)\cdot\sqrt{n}\right) = O\left(n\sqrt{n}\right)$. אם אנחנו רוצים לפעול לפי השלב השלישי באלגוריתם וכל מה שנתון לדיון פה זו שיטת המיון שלנו, מה שהיינו רוצים להראות זה שכל מיון לבעיה הזאת יניב לנו זמן ריצה של $\boldsymbol{\Omega\left(n\sqrt{n}\right)}$ (כלומר, לפחות $n \cdot \sqrt{n}$).

נביט בדוגמה הבאה:

השאילתה הראשונה הין בין הטווח $\left[0,\sqrt{n}\right]$, השאילתה השנייה היא בין $\left[0,2\cdot\sqrt{n}\right]$ וכן הלאה עד לטווח $\left[0,\left(\sqrt{n}-1\right)\sqrt{n}\right]$. השאילתה הבאה היא בין הטווח $\left[\sqrt{n},2\cdot\sqrt{n}\right]$, זו שאחריה בטווח $\left[\sqrt{n},3\cdot\sqrt{n}\right]$ וכן הלאה. כלומר, מכל בלוק יש שאילתה מהאינדקס הראשון שלו עד האינדקס הראשון של כל בלוק אחריו. כמה שאילתות יש לנו? מתוך $\sqrt{n}$ אפשרויות לבחור שני אינדקסים, מקום שמאלי ל-i ומיקום מיני ל-j, לכן יש לנו $\binom{\sqrt{n}}{2}$ אפשרויות בחירה, שזה $\boldsymbol{\Theta(n)}$ **שאילתות**.

נזכר שאנחנו רוצים להראות כי המיון שלנו מניב את זמן הריצה הטוב ביותר עבור הדוגמה לעיל.

הרעיון מאחורי הדוגמה: משום שהאלגוריתם, בלי קשר למיון, מתקדם משאילתה אחת לשאילתה עוקבת הוא עושה אחד משני הדברים בדוגמה הזו:

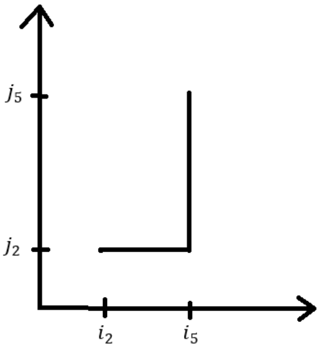
1. מזיז את L לפחות $\sqrt{n}$ צעדים ימינה.

2. מזיז את R לפחות $\sqrt{n}$ צעדים ימינה או לפחות $\sqrt{n}$ צעדים שמאלה.

וזה משום שהמרחק בין כל שתי שאילתות הוא או לפחות $\sqrt{n}$ תאים ימינה או לפחות $\sqrt{n}$ שמאלה. אז אנחנו יודעים שיש בסך הכל $m = \Theta(n)$ שאילתות ולא משנה מה המיון שאנחנו עושים, אם בשלב השלישי אנחנו רוצים להתקדם משאילתה אל העוקבת אליה הצעד הזה יעלה לפחות $\sqrt{n}$, כלומר $\Omega(\sqrt{n})$. בסך הכל, עבור כל מיון נקבל זמן ריצה של $m \cdot \Omega(\sqrt{n}) = \boldsymbol{\Omega(n\sqrt{n})}$.

לכן, על הקלט הזה, האלגוריתם שלנו מניב את המיון הכי טוב שאפשר לבצע.

אפשר גם לשאול את עצמנו האם עבור מקרים שהם לא ה-worst case אפשר למצוא מיון טוב יותר אך התשובה היא שאנחנו לא יודעים. הבעיה של מציאת מיון אופטימלי עבור אלגוריתם Mo בהינתן שאילתות היא בעיה np-קשה משום שהיא קשורה לבעיית הסוכן הנוסע.



<u>הסבר</u>: נביט על תזוזת הפוינטרים בין שאילתה 2 ל-5, למשל, אז המרחק שהיא עושה הוא $|i_2 - i_5| + |j_2 - j_5|$. נחשוב על ההתקדמות הזאת כהתקדמות בשני צירים באמצעות מרחק מנהטן.

אנחנו מחפשים את המיון האופטימלי, אז אם אנחנו מתחילים מ-(i_2, j_2) אנחנו רוצים בעצם מנסים להבין לאיזה נקודה הכי כדאי לעבור וממנה לאיזו נקודה הכי כדאי לעבור וכו'. אנחנ מחפשים מסלול אופטימלי, קצר ביותר, בגרף לפי מרחק מנהטן. מציאת מסלול קצר ביותר שעוברת בכל נקודה בדיוק פעם אחת זו בעיית הסוכן הנוסע (זה מקרה ספציפי שלה) והיא np-קשה.

4. <u>המרכיבים הייחודיים בסכום טבעי</u> (<u>לא העשרה!</u>)

(זו דוגמה נוספת לסכמה שמניבה אלגוריתמים שפועלים בזמן ריצה שכולל שורש של גודל הקלט).

הטכניקה שימושית באלגוריתמים המבצעים סריקה של סדרת איברים טבעיים וניתן להחליפה בסריקת האיברים הייחודיים בסדרה זו.

דוגמא: 1,2,2,7

במקום לסרוק מימין לשמאל את המערך, לפעמים מספיק לסרוק רק את 1,2 ו-7 ונוכל לטפל באיבר הייחודי 2 בפחות עבודה (במקום בשתי איטרציות באיטרציה אחת) ובהכללה אנחנו יכולים להקטין משמעותית את מספר האיטרציות.

<u>טענה:</u>

בהינתן סדרת טבעיים $a_1, \ldots, a_n$ שסכומה K ובה $t \leq n$ איברים יחודיים, מתקיים

$$\boldsymbol{t = O(\sqrt{K}) = O\left(\sqrt{\sum_{i=0}^{n} a_i}\right)}.$$

ובמילים: מספר האיברים השונים בסדרה קטן או שווה אסימפטוטית לשורש סכום האיברים.

עדיין לא ברור למה הדבר הזה יותר זול מסתם פשוט n, אבל יש מקרים בהם זה עובד.

אם ניתחנו כבר את גודל הקלט במונחי K אז באמצעות הטענה נצליח לנתח במונחי $\sqrt{K}$ – זה תלוי דוגמה ולא בכל מקרה זה יניב תוצאה שהיא אסימפטוטית טובה יותר.

הוכחה:

$$\mathbf{K} = \sum_{i=1}^{n} a_i \geq \sum_{i=1}^{t} a_i \geq \sum_{i=1}^{t} i = \Theta(t^2) \;\Rightarrow\; K = \Omega(t^2) \Rightarrow t = O\left(\sqrt{K}\right)$$

הערות:

- במעבר הראשון אנחנו אומרים שסכום כל איברי הסדרה ≤ סכום t האיברים הראשונים שלה ($t \leq n$).
- במעבר השני אנחנו מניחים, בלי הגבלת הכלליות, ש-t האיברים הייחודיים בתחילת הסדרה.
- מעבר שלישי: אנחנו יודעים ש-t המספרים הראשונים שונים זה מזה וכולם טבעיים, אז הדרך להקטין את הסכום שלהם הכי הרבה היא להניח כי הראשון הוא 1 וכל מספר עוקב גדול מקודמו ב-1.
- השתמשנו ב-Θ ולא ב-O כי צריך את ה-Ω בשביל הגרירה.

נראה דוגמה קונקרטית בה באמצעות הטענה נראה שסריקת איברים ייחודיים עדיפה על פני מעבר נאיבי על כל המערך:

דוגמא: בהינתן סדרת מטבעות שערכיהם טבעיים, מעוניינים לחשב את מספר הסכומים האפשריים שניתן לייצר מהם.

נחזור לדוגמא הקודמת שלנו:

נתון לנו מטבע של שקל, שני מטבעות של שנקל ומטבע חדש של 7 שקלים:

1,2,2,7

הסכומים השונים שניתן לקבל הם: 0,1,2,7,3,4,9,5,10,11,12.

כבר אנחנו יכולים לראות שכל אלגוריתם שיפתור את הבעיה הזו לא יעשה זאת ב-$O(n)$ או בפונקציה שהיא פולינומית ב-n, כי אפילו רק כמות הסכומים האפשרית גדולה אסימפטוטית בכל פונקציה פולינומית ב-n.

אבל יכול להיות שהאלגוריתם יעבוד בזמן ריצה שכולל את ערך המטבע הכי גדול, או את סכום האיברים – זה כבר סביר ותכף נראה משהו כזה.

מרגיש טבעי להשתמש פה בתכנון דינאמי, וזה מה שנעשה.

ניסיון 1 (נאיבי, לא משתמש בערכים ייחודיים):

הגדר נוסחה שתתאים לטבלת התכנון הדינאמי:

$$\boldsymbol{f(i,j)} = \begin{cases} \mathbf{T} & \text{ניתן להגיע לסכום } \boldsymbol{j} \text{ עם } \boldsymbol{i} \text{ המטבעות הראשונים} \\ \mathbf{F} & \text{אחרת} \end{cases}$$

נגדיר את הטבלה כך:

$$\boxed{\boldsymbol{f(i,j)} = \begin{cases} \mathbf{T} & \mathbf{i = 0 \ \wedge \ j = 0} \\ \mathbf{F} & \mathbf{i = 0 \ \wedge \ j > 0} \\ \boldsymbol{f(i-1,j) \vee f(i-1,j-k_i)} & \boldsymbol{i > 0} \end{cases}}$$

הסבר למקרה שאינו מקרי הבסיס:

אנחנו רוצים לדעת האם אפשר להגיע לסכום j באמצעות i המטבעות הראשונים.

כשממלאים תא נתון אנחנו מסתכלים על המטבע ה-i בסדרה. יש לנו שתי אפשרויות:

1. אפשר להגיע לסכום j גם בלי המטבע ה-i, כלומר $f(i-1,j) = T$, ואז בטוח אפשר להגיע אליו באמצעות i המטבעות הראשונים.
2. אפשר להגיע ללא המטבע ה-i לסכום j פחות הערך k_i של המטבע ה-i ואז אם נוסיף את המטבע ה-i נוכל להגיע לסכום j. כלומר, $f(i-1,j-k_i) = T$.

סכמה של מילוי כל את בטבלה:

מחפשים בתא למעלה ובשורה למעלה בתא מסוים משמאל

לבסוף, החזר את מספר ערכי ה-T בשורה האחרונה – זו תהיה כמות כל הסכומים אליהם אפשר להגיע בעזרת כל המטבעות שיש.

זמן הריצה של ניסיון 1:

כמות השורות היא כמות המטבעות ואת כמות העמודות נסמן ב-K כי אין טעם להסתכל על סכומים שאי אפשר להגיע אליהם. אנחנו ממלאים כל תא ב-$O(1)$ ולכן זמן הריצה הוא:

$$\boldsymbol{O(n \cdot K) = O\left(n \sum_{i=0}^{n} a_i\right)}$$

ניסיון 2:

הגדר מערך A עם ערכי המטבעות הייחודיים.

בנוסף, הגדר מערך B כך שמתקיים: מספר ההופעות של הסוג ה $\boldsymbol{A[i]}$ – בסדרה $= \mathbf{B[i]}$.

למשל, בדוגמה שלנו:

$$B = \begin{array}{|c|c|c|}\hline 1 & 2 & 1 \\ \hline\end{array}$$

$$A = \begin{array}{|c|c|c|}\hline 1 & 2 & 7 \\ \hline\end{array}$$

כל תא בטבלה שלנו יכיל שני דברים:

1. ערך T/F – מציין האם ניתן להגיע לסכום j באמצעות i **הסוגים** הראשונים.

   למשל, בדוגמה שלנו $f(2,5)$ יכיל T כי אפשר להגיע עם שני שנקלים ושקל אחד ל-5.

2. ערך * מסוים – בכמה מטבעות מהסוג ה-i השתמשנו.

$$F(i,j) = \boxed{\frac{\mathrm{T/F}}{*}}$$

$$\boldsymbol{f(i,j)} = \begin{cases}
\boxed{\frac{\mathbf{T}}{\mathbf{0}}} & \mathbf{i = 0 \wedge j = 0} \\[2ex]
\boxed{\frac{\mathbf{F}}{\mathbf{0}}} & \mathbf{i = 0 \wedge j > 0} \\[2ex]
\boxed{\frac{\mathbf{T}}{\mathbf{0}}} & \boldsymbol{f(i-1,j)} = \boxed{\frac{\mathbf{T}}{*}} \\[2ex]
\boxed{\frac{\mathbf{T}}{\mathbf{1}}} & \left(\begin{array}{c}\boldsymbol{f(i-1,j)} = \boxed{\frac{\mathbf{F}}{*}} \wedge \\ \boldsymbol{f(i-1,j-A[i])} = \boxed{\frac{\mathbf{T}}{*}}\end{array}\right) \\[4ex]
\boxed{\frac{\mathbf{T}}{\boldsymbol{p+1}}} & \left(\begin{array}{c}\boldsymbol{f(i-1,j)} = \boxed{\frac{\mathbf{F}}{*}} \wedge \\ \boldsymbol{f(i-1,j-A[i])} = \boxed{\frac{\mathbf{F}}{*}} \wedge \\ \boldsymbol{f(i,j-A[i])} = \boxed{\frac{\mathbf{T}}{\boldsymbol{p}}} \wedge \ (\boldsymbol{p < B[i]})\end{array}\right) \\[4ex]
\boxed{\frac{\mathbf{F}}{*}} & \boldsymbol{else}
\end{cases}$$

הסבר לגבי כל מקרה:

1. בסיס
2. בסיס
3. אפשר להגיע לסכום j עם i-1 הסוגים הראשונים, אז בוודאי אפשר עם i הסוגים הראשונים.

4. עם i-1 הסוגים הראשונים אי אפשר להגיע ל-j, אבל עם i-1 הסוגים הראשונים אפשר להגיע ל-(j-A[i]),אז אם מוסיפים את סוג המטבע ה-i בוודאי מוסיפים את ערכו ואפשר להגיע ל-j.
5. אם אי אפשר להגיע עם i-1 הסוגים הראשונים ל-j וגם אי אפשר להגיע ל-(j-A[i]) (למשל אם בדוגמה שלנו אנחנו רוצים להגיע לסכום 5 – אי אפשר להגיע ל-5 עם הסכום הראשון וגם אי אפשר להגיע אל 5-2=3), אבל עם i הסוגים הראשונים אפשר להגיע ל-(j-A[i]) וגם שיישאר לנו מטבע אחד מסוג i (למשל בדוגמה שלנו אפשר להגיע ל-3 עם מטבע אחד של 1 ומטבע אחד של 2 וגם נשאר לנו עוד מטבע של 2 להשלים ל-5) אז אפשר להגיע ל-(j-A[i]) ואז להוסיף עוד מטבע אחד מהסוג i.

סכמה של מילוי כל את בטבלה:

מחפשים בתא למעלה, בשורה למעלה בתא מסוים משמאל ובשורה הנוכחית בתא מסוים משמאל

<u>זמן ריצה של ניסיון 2</u>

עלות מילוי כל תא היא $O(1)$ ולכן זמן הריצה מורכב מגודל הטבלה + עלות המיון.

- מיון בעלות של $O(n \cdot log(n))$ – נחוץ על מנת להרכיב את המערכים A ו-B.
- גודל הטבלה הוא $O(t \cdot K) \leq O(\sqrt{K} \cdot K)$.

בסך הכל $O(n \cdot log(n) + t \cdot K) \leq O\left(n \cdot log(n) + \sqrt{K} \cdot K\right) = \boxed{\boldsymbol{O\left(\sqrt{K} \cdot K\right)}}$

החסם ההדוק ביותר יהיה $O(n \cdot log(n) + t \cdot K)$

<u>מסקנות לגבי שני הפתרונות:</u>

ניסיון 1 עבד ב-$O(n \cdot K)$ ויש קלטים עבורם זה יוצא $O(K^2)$ – למשל אם כל המטבעות שווים 1.

זמן הריצה של ניסיון 2 הוא תמיד $O(\sqrt{K} \cdot K)$, שזה קטן מ-$O(K^2)$.

כלומר, זמן הריצה של ניסיון 2 **תמיד** קטן יותר מה-worst cast של ניסיון 1.

בנוסף, גם אם לא מסתכלים על ה-worst case של ניסיון 1, כלומר על זמן ריצה של$O(n \cdot K)$, אז זמן הריצה של ניסיון 2 הוא $O(t \cdot K + n \cdot log(n))$ ומשום שמתקיים $t \leq n$ וגם $n \cdot K \leq n \cdot log(n)$ אז ניסיון 2 לא יהיה איטי יותר.

לסיכום – זה נכון שיש קלטים עבורם זמן הריצה של שני הפתרונות זהים (אם כל המטבעות שונים), אך יש קלטים עבורם פתרון 2 מהיר יותר. לכן אנחנו חושבים על פתרון 2 כעדיף.

בעיית ה-*RMQ - $\boldsymbol{(O(n), O(1), O(n))}$:

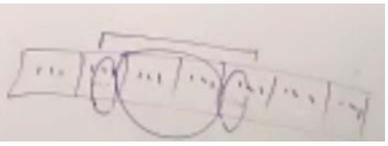
זהה להגדרת RMQ המקורית, למעט ההנחה כי ההפרש שבין כל זוג איברים עוקבים הוא 1 או -1.

ניסיון 1:

- **עיבוד מוקדם:**

1. חלק את המערך לבלוקים בגודל $\mathrm{L} = \frac{1}{2}\log(\mathrm{n})$ וחשב מערך B של ערכי מינימום המתאימים לאוסף הבלוקים.
2. הגדר את העיבוד המוקדם של RMQ4 על B.
3. הגדר עיבוד מוקדם של RMQ4 על כל בלוק.

- **מענה על שאילתה:**



1. הבע את הקטע כרצף בלוקים ועוד איברי "זנב" וחשב את המינימום של רצף הבלוקים בעזרת המערך B. **עלות:** $\boldsymbol{O(1)}$
2. חוץ מזה, חשב את המינימום של הזבנות עם RMQ4. **עלות:** $\boldsymbol{O(1)}$
3. החזר את האיבר המינימלי מבין שלושת האיברים שהתקבלו. **עלות:** $\boldsymbol{O(1)}$

זמן הריצה של ניסיון 1 - $\boldsymbol{(O(n \cdot log(\log(n))), O(1), O(n \cdot log(\log(n))))}$:

- **עיבוד מקדים:** $\boldsymbol{O(1)}$ – פירוט לעיל.
- **מענה על שאילתה:**

1. חישוב log הוא לא יקר ויחד עם מציאת מינימום בכל הבלוקים העלות היא $O(n)$.

2. עלות: באופן כללי העיבוד המקדים של RMQ4 עלה $O(m \cdot log(m))$ כאשר m הוא גודל המערך. אנחנו יודעים שכל בלוק הוא בגודל L ולכן מספר הבלוקים במערך B הוא $\frac{n}{L}$ ולכן עלות העיבוד המוקדם של RMQ4 על B היא:

$$O(\frac{n}{L} \cdot log(\frac{n}{L})) = O(\frac{n}{\frac{1}{2}\log(\mathrm{n})} \cdot log(\frac{n}{\frac{1}{2}\log(\mathrm{n})})) \leq O(\frac{n}{\frac{1}{2}\log(\mathrm{n})} \cdot log(n)) = O(n)$$

הסיבה שבחרנו את גודל הבלוקים להיות $\Theta(log(n))$ היא כדי שהערך הזה יצטמטם עם ה-Log השני בביטוי.

3. כאמור יש לנו $\frac{n}{L}$ בלוקים, זו כמות הפעמים שהפעלנו עיבוד מקדים של RMQ4 בשלב זה – כל פעם על בלוק בגודל L. לכן עלות השלב היא:

$$O(\frac{n}{L} \cdot L \cdot log(L)) = O(n \cdot log(\frac{1}{2}\log(\mathrm{n}))) = \boldsymbol{O(n \cdot log(\log(n)))}$$

קיבלנו פתרון שעולה לנו $\boxed{\left(O(n\cdot log(\log(n))), O(1), O(n\cdot log(\log(n)))\right)}$.

בינתיים לא השתמשנו עדיין בהנחה של פלוס מינוס אחד וגם לא בחצי של L.

ניסיון 2:

לכל בלוק A במערך המקורי, נגדיר את הבלוק הקנוני D המתאים לו כבלוק עם D[0]=0 וסדרת הפרשיו זהה לזו של A.

דוגמה:

A =

3	4	3

D =

0	1	0

<u>הבחנות:</u>

1. מספר הבלוקים הקנוניים הוא 2^{L-1}

(האיבר הראשון הוא אפס ולכל איבר אחר 2 אפשרויות: גדול מקודמו ב-1 או קטן ממנו ב-1).

2. לכל i מתקיים כי $\mathrm{A[i]} = \mathrm{D[i]} + \mathrm{A[0]} \Leftarrow min\{A[i,\dots,j]\} = min\{D[i,\dots,j]\} + A[0]$.

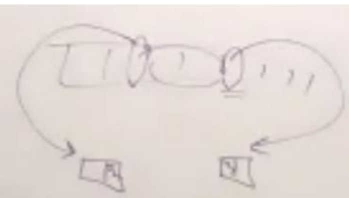
- **עיבוד מוקדם:**

נתקן את העיבוד המוקדם שלנו מניסיון 1 כך שבמקום <u>שלב 3</u> נכתוב:

א. נגדיר לכל בלוק במערך המקור את הבלוק הקנוני המתאים לו.

ב. הגדר RMQ4 על כל בלוק קנוני.

מה שהפריע לנו קודם זה שהרצה של RMQ4 על הרבה בלוקים היא יקרה, עכשיו צמצמנו את מספר הבלוקים עליהם אנחנו עושים RMQ4 (כי, כאמור, מספרם מוגבל).



- **מענה על שאילתה:**

זהה לניסיון 1 מלבד לחישוב המינימום בזנבות - שם נשתמש בהפניה לבלוקים הקנוניים.

<u>**זמן ריצה של ניסיון 2 - $\left(O(n), O(1), O(n)\right)$:**</u>

- **מענה על שאילתה - $O(1)$**

אנחנו מוצאים את המינימום על רוב הקטע ב-$O(1)$ בעזרת B, מוצאים את הבלוקים הקנוניים המתאימים ב-$O(1)$, הגדרנו RMQ4 על הבלוקים הקנוניים אז מציאת המינימום עליהם עולה $O(1)$ ולבסוף מציאת מינימום על 3 איברים עולה $O(1)$.

- **עיבוד מוקדם –**

שינינו רק את שלב 3:

א. $\boldsymbol{O(n)}$: עבור כל בלוק - רצים על הבלוק, עוקבים אחר ההפרשים של 1 או 1- ולפי זה מוצאים את הבלוק הקנוני המתאים.

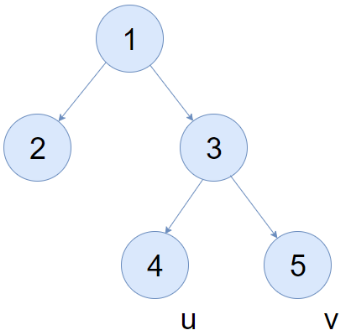
ב. $\boldsymbol{O(n)}$: יש 2^{L-1} בלוקים קאנונים ועל כל אחד מצבעים RMQ4:

$$O(2^{L-1}\cdot L\cdot log(L)) \le O(2^{L}\cdot L\cdot log(L)) = O(2^{\frac{1}{2}\log(n)}\cdot L\cdot log(L))$$
$$= O(\sqrt{n}\cdot log(n)\cdot log(log(n))) \le \boldsymbol{O(n)}$$

בסך הכל, שלב 3 עולה לנו $\boldsymbol{O(n)}$.

וזה לעומת עלות שלב 3 בניסיון 1, $O(n\cdot log(\log(n)))$.

<u>LCA5:</u>



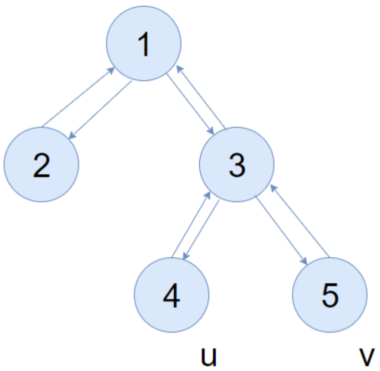
נרצה לתרגם את העץ למערך ע"י סריקתו. נרצה שהסריקה תקיים מספר תכונות:

1. בסריקת הקטע בין u ל- v במערך לא נמצא קודקוד גבוה מה-LCA שלהם.
2. בקטע בין u ל-v במערך יופיע ה-LCA שלהם.

אז אם מוצאים אב קדמון משותף במערך, בוודאות הוא ה-LCA כי אין אף קודקוד, ובפרט אף אב קדמון, אחר עם רמה נמוכה יותר.

3. ה-LCA מתאים לשאילתת מינימום (המינימום יהיה על רמת הקודקודים).

<u>הגדרה – מסלול אוילר:</u> מסלול שעובר בכל צלע בדיוק פעם אחת.



- **עיבוד מוקדם:**

1. נגדיר שני מערכים, E^1 ו-E^2 באופן הבא:
   - נעבור על העץ בצורה של מסלול אוילר באמצעות DFS מהשורש.
   - בכל פעם שנגיע לקודקוד נוסיף אותו ל-E^1, נתעד ב-E^2 את רמת הקודקוד.
2. נגדיר מערך $\bar{E}$ שישמור לכל קודקוד את אינדקס הופעתו הראשונה ב-E^1.

דוגמא המתאימה לגרף הדוגמא משמאל:

$E^1 =$

				v		u		
1	2	1	3	4	3	5	3	1

$E^2 =$

0	1	0	1	2	1	2	1	0

$\bar{E} =$

1	2	3	4	5
1	2	4	5	7

3. נגדיר עיבוד מוקדם של RMQ* על המערך E^2. מותר לנו להגדיר את העיבוד הזה על E^2 כי ההפרשים בין איברים עוקבים בו הם רק ± 1.

- **מענה על שאילתה:**

החזר את:

$$E^1[\text{argminRMQ}^*_{E_2}[\bar{E}[v], \bar{E}[u]]]$$

- $\bar{E}[v]$ – האינדקס של v ב-E^1 ו-E^2.
- $\bar{E}[u]$ - – האינדקס של u ב-E^1 ו-E^2.
- $\text{argminRMQ}^*_{E_2}[\bar{E}[v], \bar{E}[u]]$ – מחפשים את הקודקוד עם הדרגה המינימלית בקטע שבין v ל-u ב-E^1, לכן מחפשים מינימום על הקטע $[\bar{E}[v], \bar{E}[u]]$ ב-E^2 באמצעות RMQ*. כאשר מוצאים את המינימום, אנחנו בעצם מעוניינים באינדקס שלו בלבד (כדי למצוא את הקודקוד ב-E^1), לכן יש בביטוי argmin.
- $E^1[\text{argminRMQ}^*_{E_2}[\bar{E}[v], \bar{E}[u]]]$ – הקודקוד עם הדרגה המינימלית בקטע בין v ל-u ב-E^1.

ניתוח זמן ריצה:

- **עיבוד מוקדם:**

1. בניית המערכים היא בעלות סריקת העץ - $O(n)$.
2. אפשר להגדיר את $\bar{E}$ כך: עוברים על E^1 ואם זו הפעם הראשונה שביקרנו בו, מתעדים את האינדקס של E^1 ב-$\bar{E}$.
3. גודל E^1 ו-E^2 הוא 2n-1 כל אחד[6], לכן עיבוד מוקדם של RMQ* על E^2 יעלה לנו $O(n)$.

בסך הכל, העיבוד המוקדם עולה לנו $O(n)$.

- **שאילתה: $O(1)$.**

רעיון נכונות הפתרון:

מימשנו סריקת מסלול אוילר באמצעות DFS, לכן במסלול שבין שני קודקודים לא יהיה לנו אף קודקוד גבוה מה-LCA.בנוסף, אין בין u ל-v קודקודים שאינם בתת-עץ ששורשו ה-LCA בזכות ה-DFS. חוץ מזה, ה-LCA בטוח מופיע בין שני הקודקודים משום שניתן להוכיח טענה כללית לגבי עצים לפיה כל מסלול בין שני קודקודים בהכרח עובר דרך ה-LCA שלהם.

לכן, ה-LCA הוא הקודקוד הכי הגבוה בעץ, כלומר בעל הדרגה הנמוכה ביותר, בקטע ב-E^1 שבין שני הקודקודים.

---

[6] בעץ יש n-1 צלעות. הכפלנו כל צלע, אז יש 2n-2 צלעות בגרף. מתעדים את השורש במערך ולאורך התקדמות ה-DFS מתעדים כל קודקוד שנמצא בסוף צלע שהגענו אליה, לכן יש 2n-1 תיעודי קודקודים.

שיעור 5 - 16.11.2020

עצי קטעים

עץ קטעים (לשאילתות):

מבנה נתונים דינאמי (תומך בעדכונים באופן יעיל) המייצג סדרת נתונים $a_1, \dots, a_n$

*(אפשר גם לחשוב עליה כקבוצה – אנחנו לא דורשים סדר על האיברים).

**(אנחנו נתייחס לסדרה שגודלה הוא חזקה שלמה של 2, אך גם אם לא כך הדבר זה לא יקר להוסיף לה איברים).

העץ משמש לפעולות הבאות:

1. **שאילתה (i,j):** החזר את $f(a_i, \dots, a_j)$ עבור f נתונה מראש.

   למשל, f יכולה להיות סכום ואז נאפשר לבצע שאילתא על סכום קטעים סדרה (ביעילות).

2. **עדכון (i,x):** עדכן את ערכו של a_i להיות x.

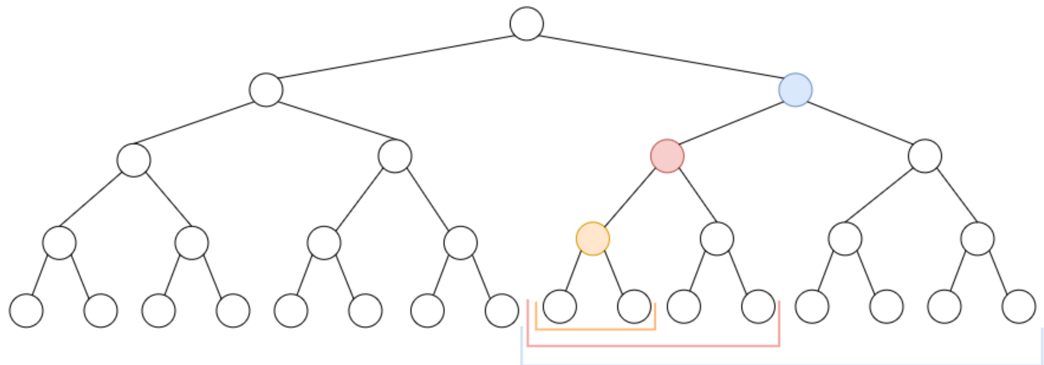
דרישות: פעולות השאילתה והעדכון בעלות $\boxed{O(log(n))}$.

מבנה הנתונים:

בהינתן סדרה $a_1, \dots, a_n$ נגידר עץ בינארי ***שלם***[7] (עץ שכל הרמות בו מלאות) בעל n עלים, כלומר, 2n-1 קודקודים. כל קודקוד בעץ "שולט" על קטע רצוף במערך.

למשל, אם f היא סכום אז קודקוד ש"שולט" על קטע במערך מחזיק את סכום הקטע.

להלן דוגמה בה מסומנים 3 קודקודים ב-3 צבעים שונים וקטעי השליטה שלהם בצבעים המתאימים:



הדרישה מ-f היא שהיא תהיה מוגדרת לפי אופרטור בינארי אסוציאטיבי.

בינארי: קיים אופרטור $\circ$ המוגדר כך: $\circ : \tilde{A} \times \tilde{A} \to \tilde{A}$ (למשל חיבור בין טבעיים שמחזיר מספר טבעי) כך שמתקיים $f(a_i, \dots, a_j) = a_i \circ a_{i+1} \circ \dots \circ a_j$.

---

[7] זה **לא** אותו הדבר כמו עץ שלם! מתוך ויקיפדיה:

- **עץ בינארי מלא** הוא עץ בו לכל צומת שאינו עלה יש שני בנים.
- **עץ בינארי מושלם** (נקרא גם "עץ שלם") הוא עץ בינארי מלא, בו כל העלים הם מאותה רמה.

אסוציאטיבי: $\forall x,y,z \in \tilde{A} \quad (x \circ y) \circ z = x \circ (y \circ z)$

***נשים לב שאין דרישה שהאופרטור יהיה קומוטטיבי ולכן, אפריורית, אסור לנו להחליף את סדר הארגומנטים של f.**

יש אופרטורים שהם במקרה קומוטטיביים (כמו חיבור) ואז זה בסדר לכתוב $f(a_2, a_1 \dots, a_n)$ במקום $f(a_1, a_2 \dots, a_n)$, אבל עבור כפל מטריצות, למשל, זה לא עובד.

דוגמאות לאופרטורים חוקיים:

כפל, חיבור, כפל מטריצות, הרכבת פונקציות (ואז בעלים יש פונקציות), gcd.

מימוש הפעולות – כדי להקל על הכתיבה עוברים לעבוד על מקרה פרטי בו f היא סכום:

אתחול (A):

הגדר עץ בינארי שלם בעל $|A|$ עלים, כלומר, $2|A| - 1$ קודקודים.

כל קודקוד v יחזיק את השדות:

1. **v.parent, v.left, v.right** – בניו ואביו.
2. **v.a ו-v.b** – הקודקוד השמאלי ביותר והימני ביותר בקטע השליטה של v, בהתאמה.
3. **v.value** – ישמור את:

$$\sum_{\substack{leaf_i \text{ is a leaf} \\ \text{in the sub-tree} \\ \text{whose root is } v}} leaf_i.value$$

חישוב הערך מתקבל באמצעות:

$$v.value = v.left.value + v.right.value$$

בנוסף, הגדר מערך P כך ש-P[i] מצביע לעלה ה-i.

P מאפשר לנו גישה ב-$O(1)$ לעלה מסוים, במקום להשתמש בחיפוש בינארי החל מהשורש.

זה שימושי במקרים בהם אנחנו רוצים לתמוך בהרחבות לעץ.

עלות האתחול היא $\boldsymbol{O(n)}$:

את כל השדות אפשר למלא באמצעות שימוש ב-DFS בעלות של $O(n)$ וכן אתחול P עולה $O(n)$.

עדכון (i,x):

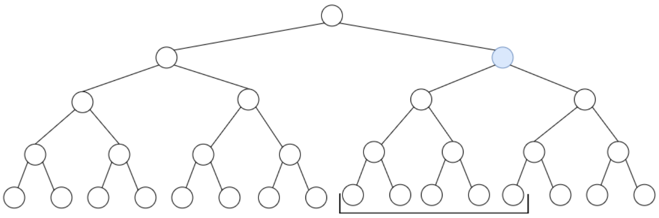
הגדר את P[i] (כלומר, את מי ש-P[i] מצביע עליו) להיות x ולכל אב קדמון שלו (החל מהעלה כלפי מעלה), v, עדכן: $\boldsymbol{v.value = v.left.value + v.right.value}$.

זמן ריצת העדכון - $\boxed{O(log(n))}$ **:** החלפת P[i] עולה $O(1)$. עדכון כל אב קדמון יעשה ב-$O(1)$ (כי יש לנו שדה v.parent) ויש לכל עלה $O(h) = O(log(n))$ אבות קדמונים.

<u>שאילתה (i, j):</u>

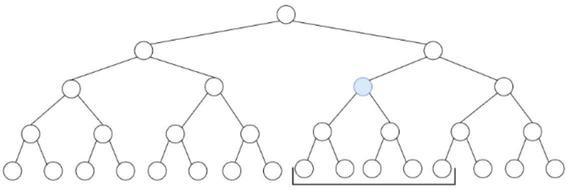
הגדרה – "קודקוד הפיצול": הקודקוד הנמוך ביותר שקטע השליטה שלו מכיל במלואו $[i, j]$.

למשל, בדוגמא למטה, הקודקוד התכלת הוא קודקוד הפיצול עבור שאילתה על הקטע המסומן:

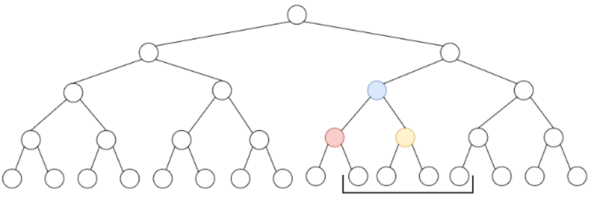


1. התקדם מהשורש עד למציאת **קודקוד הפיצול**, v (באמצעות בדיקת i ו-j למול ערכי a ו-b).
2. חשב את ערך (סכום במקרה שלנו) **הסיפא** של $v.left$ שמתחילה ב-i באופן הבא:
   - כל עוד $v \neq null$:
     א. אם $i = v.a$, הוסף את v.val לסכום וסיים.

     למשל, עבור השאילתה על הקטע המסומן, כשנגיע לקודקוד הכחול אפשר להחזיר את ה-value שלו.

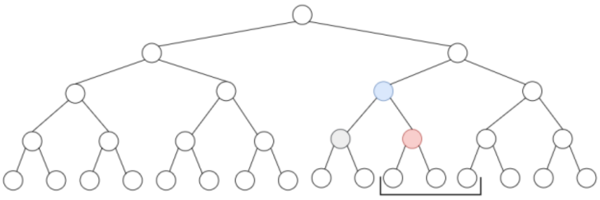


     ב. אם i מוכל בקטע השליטה של v.left, הוסף את v.right.val לסכום ועדכן את v להיות v.left. למשל, בדוגמא משמאל, כשנגיע במהלך חישוב הסיפא לקודקוד הכחול נוסיף את ה-value של הקודקוד הצהוב לסכום ונמשיך לורוד.



     ג. אחרת, עדכן את v להיות v.right.

     כלומר, מתעלמים מהתת-עץ השמאלי. בדוגמה משמאל, כשנגיע לקודקוד הכחול, נמשיך אל בנו הימני הורוד ונוותר לגמרי על תת העץ ששורשו הוא הבן השמאלי האפור.



3. חשב את **הרישא** של v.right שמסתיימת ב-j.
   חישוב הרישא סימטרי לחישוב הסיפא.

זמן ריצת המענה על שאילתה - $\boxed{O(log(n))}$ **:**

1. מציאת קודקוד הפיצול בעלות חיפוש בינארי - $O(log(n))$.

2. מציאת הקודקודים הרלוונטיים בעלות גובה העץ – $O(log(n))$.

בכל רמה במהלך שלב 2 לוקחים את ה-val רק של קודקוד אחד (הנוכחי או אחד הבנים) וממשיכים לרמה הבאה. יש $O(log(n))$ רמות, עוברים על כולן במקרה הגרוע ובכל רמה מבצעים פעולה.

<u>עץ קטעים דואלי (לעדכונים):</u>

משמש לייצוג סדרת איברים ותומך בפעולות הבאות:

1. <u>שאילתה (i):</u> החזר את a_i.
2. <u>עדכון (i,j,x):</u> לכל $i \leq t \leq j$ עדכן את a_t להיות $x \circ a_t$

(במקרה הפרטי של סכום – נוסיף לכל איבר בטווח $[i,j]$ את x. ש-x הוא **משמאל** לאופרטור)

<u>תכונת עץ הקטעים הדואלי:</u>

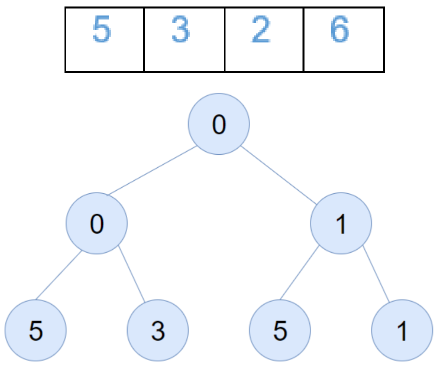
ערכו של כל איבר a_i מתקבל ע"י חישוב:

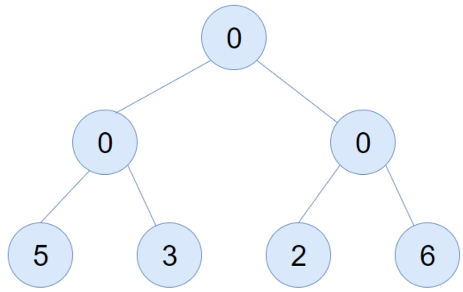
$$v_k.val \circ v_{k-1}.val \circ \ldots \circ v_1.val$$

כאשר v_1 הוא העלה המתאים ל-a_i (שזה אותו הדבר כמו P[i]) ו-v_k הוא שורש העץ,

כלומר, $v_k, v_{k-1}, \ldots, v_1$ הוא המסלול מהשורש עד לעלה.

דוגמה: להלן שני עצי קטעים דואליים אפשריים (יש יותר מאחד) עבור:





לעומת זאת,

<u>תכונת עץ הקטעים לשאילתות (עבור המקרה הפרטי של סכום):</u>

$$\sum_{\substack{leaf_i\ is\ a\ leaf\\ in\ the\ sub-tree\\ whose\ root\ is\ v}} leaf_i.value$$

<u>מימוש הפעולות:</u>

1. <u>אתחול(A):</u>

זהה לאתחול עץ קטעים לשאילתות, למעט הגדרת ה-values, הערך של העלה ה-i יהיה a_i ושל היתר יהיה 0 (או במקרה הכללי, איבר יחידה כלשהו id – למשל מטריצת הזהות עבור כפל מטריצות, פונקציית הזהות עבור הרכבת פונקציות וכו').

2. **<u>שאילתה (i):</u>**

מחשבים את הרכבת האיברים $P[i].val \circ \dots \circ v_{k-1}.val \circ v_k.val$, כאשר זהו המסלול מהשורש אל העלה המוצבע ע"י P[i].

3. **<u>עדכון (i,j,x):</u>**

<u>ניסיון 1:</u>

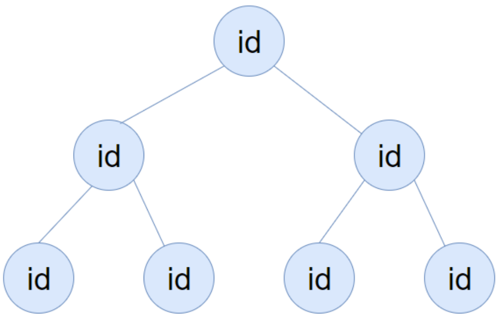
זהה לפעולת השאילתה בעץ קטעים רגיל. כלומר, מוצאים את הקודקודים הרלוונטיים ולכולם מרכיבים את x משמאל.

ניסיון 1 עובד עבור אופרטורים קומוטטיביים, אך לא במקרה הכללי. למשל:

אם איברי המערך A הן פונקציות, נתון לנו העץ הבא והפעולות הבאות אחת אחרי השנייה:

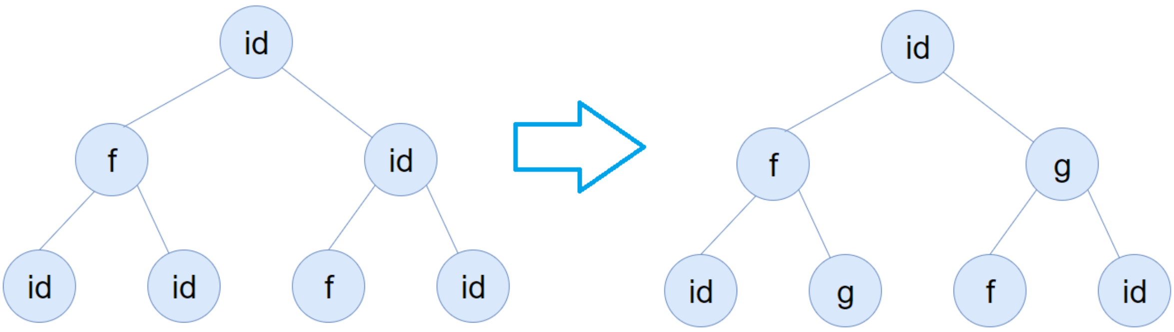
$(1,3,f)$

$(2,4,g)$



נשים לב שעבור העלים השני והשלישי משמאל אנחנו אמורים לקבל $g \circ f$.

נפעיל את f על הקודקודים הרלוונטיים ואז את g:

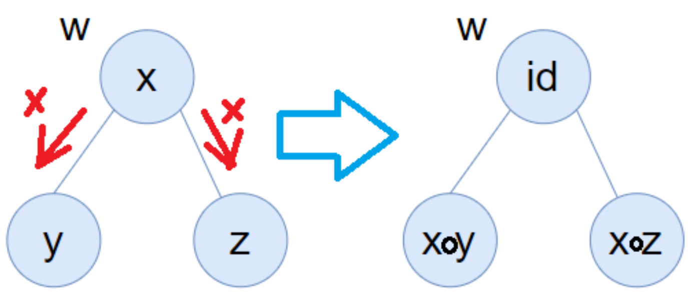


אוי לא! זה לא עבד! אמנם עבור העלה השלישי משמאל נקבל $f \circ g$ אם נעלה ממנו אל השורש ונרכיב מימין כל פונקציה שנפגוש, אבל אם נפעל באותה דרך עבור העלה השני נקבל $g \circ f$.

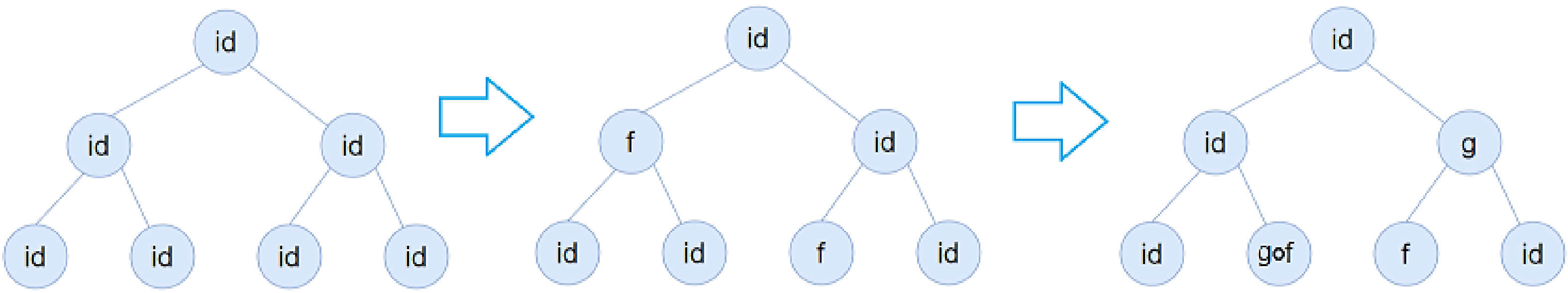
הערה: גם אם היינו מנסים לשנות רק את הדרך בה אנחנו קוראים את העץ – למשל מלמעלה למטה בתת-עץ השמאלי וההפך בימני זה עדיין לא היה עובד כי אפשר היה לקבוע שתי סדרות שונות של פעולות עדכון שמניבות את אותו העץ ואז לא נדע לפי איזו סדרה לקרוא אותו.

<u>ניסיון 2 (תומך באופרטורים לא קומוטטיביים):</u>

זהה לניסיון 1, למעט העובדה שבכל התקדמות בעץ מבצעים את פעולת "דחף מטה"(w) רק על הקודקודים המעניינים.



נבחן את השיטה החדשה על שתי הפעולות הקודמות: $(1,3,f)$ ואז $(2,4,g)$:



כעת, גם עבור עלה 2 וגם עבור עלה 3 נקבל את התוצאה $g \circ f$ כשנקרא את העץ.

שיעור 6 - 23.11.2020

פתרון תרגיל הבית

1. **נקודות ומלבנים במישור:**

נתונה סדרת n נקודות במישור $(p_1, p_2, ..., p_n)$, שהקואורדינטות שלהן הן מספרים טבעיים. כמו כן, לכל נקודה $p = (p.x, p.y)$ נתון ערך $p.val$.

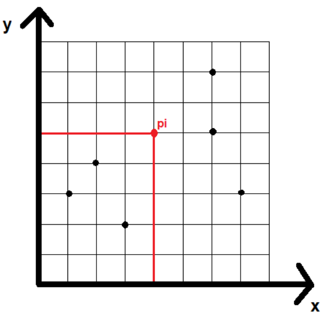
הנקודות נסרקות אחת אחרי השניה לפי הסדר בו נתונות בקלט. בתהליך זה, באיטרציה ה־i מסומנת הנקודה p_i אלא אם סומנה כבר קודם, ובהמשך מסומנת באותה איטרציה כל נקודה p_j שעדיין לא סומנה, ואשר עבורה קיימות לפחות $p_j.val$ נקודות **שכבר סומנו** (יתכן באותה איטרציה) שקואורדינטת ה־x שלהן קטנה מ־$p_j.x$ וקואורדינטת ה־y שלהן קטנה מ־$p_j.y$.

עליכם לתאר אלגוריתם שבהינתן קלט זה, מחזיר לכל נקודה את האיטרציה בה היא מסומנת במהלך התהליך המתואר.

על האלגוריתם לפעול בזמן ריצה $O(n^3)$ או טוב ממנו. הסבירו בקצרה את נכונות האלגוריתם ואת זמן ריצתו.

בונוס: תארו אלגוריתם המתבסס על הטכניקות שנלמדו עד השבוע השלישי בקורס או וריאציה שלהן, שזמן ריצתו טוב יותר מ־$O(n^2)$. אין להשתמש בפתרון במבנה הנתונים "עץ קטעים" שיילמד בהמשך הקורס או בוריאציה שלו.

פתרון ראשון בעלות $O(n^3)$



1. לכל נקודה p_i (ה-i בקלט) הגדר את זמן הסימון להיות i.
2. מיין את הנקודות לפי ערך ה-x שלהן במערך $x[1, ..., n]$ ומיפוי הפוך $\bar{x}$

($\bar{x}$ מכיל עבור כל נק' p_i את מיקומה ב-x, כלומר, את המיקום שלה לפי המיון).

3. לכל i מ-1 עד n:

א. לכל נקודה p_j מימין ל-p_i ב-x:

(אנחנו עוברים רק על הנק' שבוודאות לא קטנות מ-p_i כי רק עליהן היא עשויה להשפיע).

i. בדוק כמה נקודות משמאל ל-p_j שהן קטנות ממנה (גם בערך ה-x וגם בערך ה-y) ודלוקות יש, אם מספרן גדול מ-$p_j.val$ סמן את p_j.

4. החזר את רשימת הסימונים. **עלות:** $\boldsymbol{O(n)}$

<u>זמן ריצה - $\boldsymbol{O(n^3)}$:</u>

1. $\boldsymbol{O(n)}$.
2. $\boldsymbol{O(n \cdot log(n))}$ - עלות המיון: $O(n \cdot log(n))$, עלות בניית $\bar{x}$: $O(n)$.
3. $\boldsymbol{O(n^3)}$ - עלות הלולאה החיצונית היא $O(n)$, עלות הלולאה האמצעית היא גם $O(n)$ ועלות הלולאה הפנימית ביותר היא גם $O(n)$.

<u>נכונות (בקצרה, זו לא הוכחה):</u>

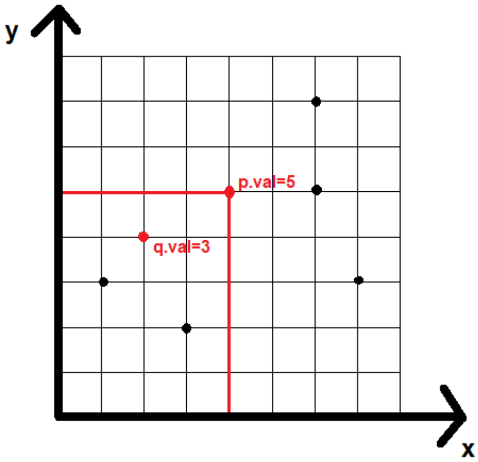
אתחלנו את ערכה של נקודה p_j במערך הפלט להיות j.

אם p_j אמורה להיות מוסמנת לפני האיטרציה ה-j, נניח באיטרציה ה-i, האלגוריתם יבדוק זאת – הוא בודק כמה נקודות מודלקות מתוך כל אלו שמשמאל ל-p_j ומתחתיה ומכיוון שאינדוקטיבית ערכן נכון אז גם ערכה של p_j.

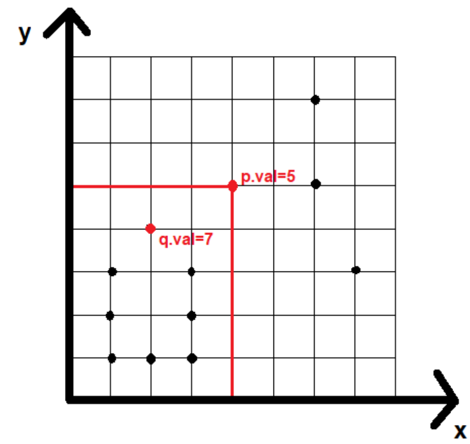
<u>פתרון שני בעלות קטנה מ-$O(n^2)$:</u>

<u>הבחנות</u>

1. ניתן לסרוק את נקודות הקלט לפי ערך ה-val שלהן:
עבור נקודה p, אם נקודה אחרת, q, רלוונטית לה ("קטנה" ממנה) אז:



- אם $q.val < p.val$ אז, אינדוקטיבית, זמן הסימון של q נכון.



- אם $q.val > p.val$, אז אם q עודכנה בעקבות נקודות כלשהן ש"קטנות" ממנה, אותן הנקודות כבר יעדכנו את p ואין צורך לעדכן את זמן הסימון של p להיות זמן הסימון המעודכן של q.

2. אם ניתן היה למיין כל רישא[8] של x לפי y וגם למיין כל רישא של אותו מערך ממוין לפי זמן סימון הנקודות – ניתן היה לקבל בקלות את זמן הסימון של הנקודה שנסרקת (רישא של כל מערך באיור משמאל הוא החלק ללא מילוי אפור כהה).
המיונים הללו יקרים ולכן נבצע רק את חלקם.

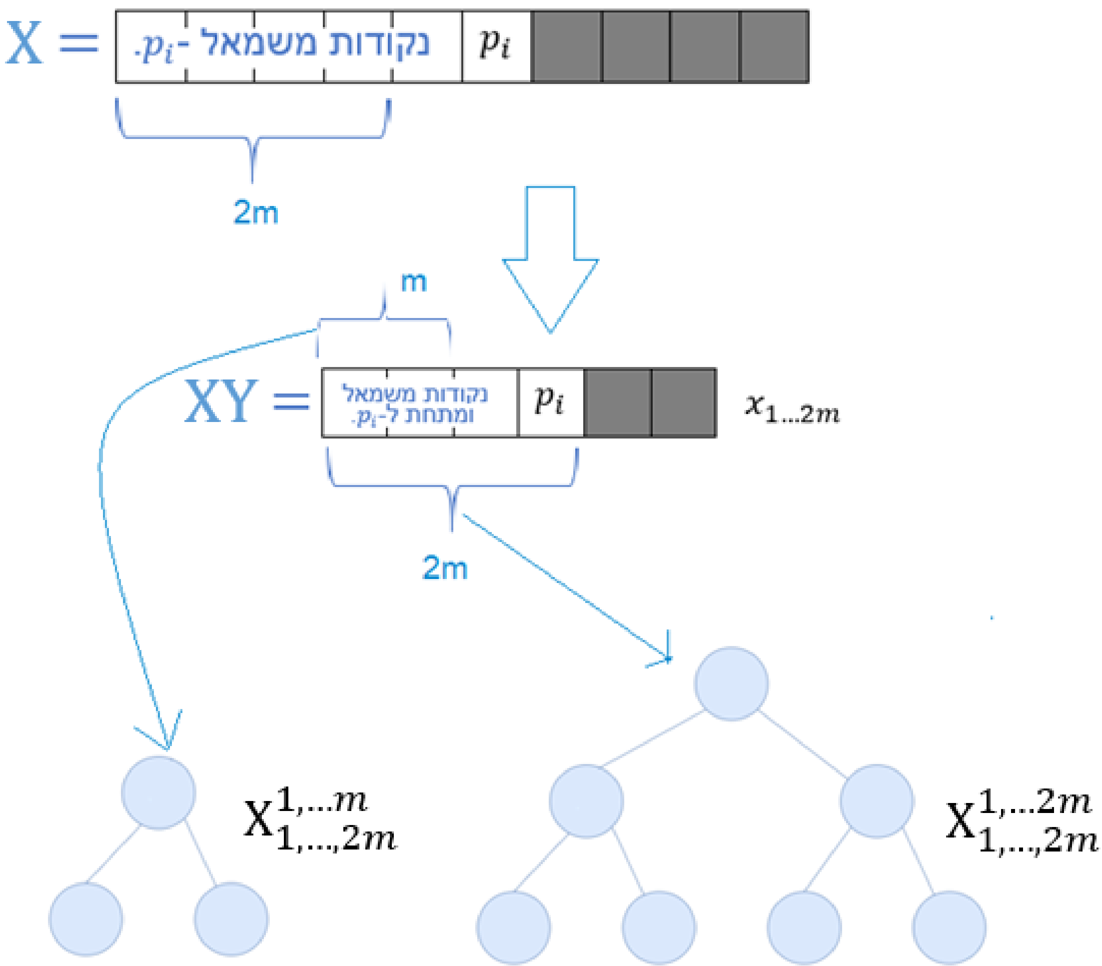
[8] "רישא" הוא החלק של המערך שמשמאל לנקודה p_i שנסרקת כרגע, באיטרציה ה-i.

אמנם אין פה ממש חלוקה לעיבוד מוקדם ומענה על שאילתה, אבל נחלק את זה באופן מלאכותי לפי חלוקה זו כאשר השאילתה היא "באיזו איטרציה הנקודה p_i נדלקה?".

עיבוד מוקדם:

1. לכל נקודה p, עדכן את איטרציית הסימון הדיפולטיבית שלה.
2. הגדר מערך X של הנקודות ממוינות לפי ערכי x וכן מיפוי הפוך $\bar{X}$.
3. נמיין לפי x ו-y את כל הרישות של X באורכים $m, 2m$ וכן הלאה, ונשמור את המערכים במערכים שנקרא להם $X_{1...m}, X_{1...2m}$ וכן הלאה. (היינו רוצים שיהיו לנו מערכים ממוינים כמו בציור לעיל עבור כל נקודה ולא רק עבור רישות באורכים קבועים, אבל זה יקר).
4. נמיין בכל אחד מהמערכים שהוגדרו בשלב 3, כל רישא באורך m,2m,... לפי זמן סימון הנקודות (תחילה לכל נקודה נמלא את הזמן הדיפולטיבי ונעדכן עם כל שינוי) ונשמור בעצי AVL $X^{1,...m}_{1,...,k\cdot m}, X^{1,...2m}_{1,...,k\cdot m}, \ldots$ ולכל עץ נשמור את קואורדינטת ה-y המקסימלית המתאימה לו.
5. מיין את הנקודות לפי ערך ה-val ושמור ב-V.

הציור הבא מתאר את התהליך עבור הרישא של x באורך 2m:



"מענה על שאילתות":

לכל i מ-1 עד n:

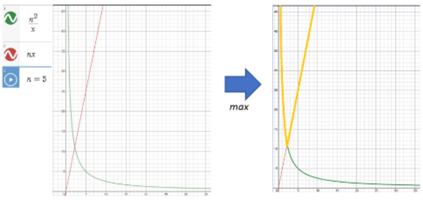
1. מצא את $V[i]$ ב-x בעזרת $\bar{X}$ ואת הרישא המקסימלית הממויינת לפי y ומתאימה לה, $X_{1\ldots km}$.
2. מצא את המיקום המתאים ל-$V[i]$ ב-$X_{1\ldots km}$ ואת עץ הרישא המקסימלית המתאימה לה שממוין לפי זמן סימון, $X_{1,\ldots,k\cdot m}^{1,\ldots t\cdot m}$.
3. הוסף נקודות מהזנבות בהתאם לצורך (= אם יש זנבות ואם יש בהם איברים עם ערך y קטן מהמקס' של העץ. למשל, בציור לעיל יש איבר מפוספס בין p_i לסוף הרישא $x_{1\ldots 2m}$ – אותה אולי נוסיף לעצים).
4. מצא את הנקודה ה-$V[i].val$ בעץ המעודכן.
5. אם יש צורך, עדכן את זמן סימון $V[i]$ במערך הפלט ובכל העצים.
6. החזר את הנקודות מסומנות.

זמן ריצה:

- **עיבוד מוקדם - $O\left(\frac{n^2}{m^2} \cdot n \cdot \log(n)\right)$:**
  1. אתחול - $O(n)$.
  2. מיון X ו-$\bar{X}$ - $O(n \cdot \log(n))$.
  3. מיון הרישות $X_{1,\ldots,km}$ - $O\left(\frac{n}{m} \cdot \log(n)\right)$.
  4. מיון העצים $X_{1,\ldots,k\cdot m}^{1,\ldots t\cdot m}$ - $O\left(\frac{n^2}{m^2} \cdot n \cdot \log(n)\right)$.
  5. מיון לפי val - $O(n \cdot \log(n))$.

- **"מענה על שאילתות" - $O\left(\left(m + \frac{n^2}{m^2}\right) \cdot \log(n) \cdot n\right)$.:**

  מעבר על v - $O(n)$ ועבור כל נקודה:
  1. $O(1)$ - מציאת $V[i]$ ב-x עולה $O(1)$ וגם מציאת הרישא עם פעולות אריטמיות.
  2. $O(\log(n))$ - מציאת $V[i]$ בעזרת ערך ה-y עם חיפוש בינארי.
  3. $O(m \cdot \log(n))$ – גודל כל זנב ב-X וב-XY הוא לכל היותר m וגודל כל עץ הוא n, אז אנחנו רוצים להוסיף $O(m)$ איברים לעץ AVL.
  4. $O(\log(n))$ – נניח שהעצים תומכים בחיפוש איבר לפי מיקום.
  5. $O\left(\frac{n^2}{m^2} \cdot \log(n)\right)$ - נחשב כמה עצים יש: ב-x יש לנו לכל היותר $\frac{n}{m}$ רישות וכל רישא כזו מכילה לכל היותר $\frac{n}{m}$ רישות (כי הרישא הגדולה ביותר של x היא בגודל n) ולכל רישא כזו בונים עץ. כלומר, יש לנו לכל היותר $\frac{n^2}{m^2}$ עצים לעדכן.



נרצה להביא למינימום את $\boldsymbol{O\left(\left(m+\frac{n^2}{m^2}\right)\cdot \log(n)\cdot n\right)}$ בעזרת m.

הגורם היחיד שתלוי ב-m הוא $\boldsymbol{\left(m+\frac{n^2}{m^2}\right)}$, אותו נרצה להביא למינימום. כפי שכבר אמרנו בשיעור 3, מינימום של O של פונקציית סכום הוא כמו מינימום של O של פונקציית המקסימום שלה, לכן נחפש את נקודת המפגש של שתי הפונקציות:

$$\boldsymbol{m=\frac{n^2}{m^2}\Rightarrow m^3=n^2\Rightarrow m=n^{\frac{2}{3}}}$$

נציב $\boldsymbol{m=n^{\frac{2}{3}}}$ ונקבל:

$$O\left(\left(n^{\frac{2}{3}}+\frac{n^2}{n^{\frac{4}{3}}}\right)\cdot\log(n)\cdot n\right)=O\left(\left(n^{\frac{2}{3}}+n^{2-\frac{4}{3}}\right)\cdot\log(n)\cdot n\right)=O\left(2n^{\frac{2}{3}}\cdot\log(n)\cdot n\right)$$

$$=\boldsymbol{O\left(n^{1\frac{2}{3}}\cdot\log(n)\right)}<O(n^2)$$

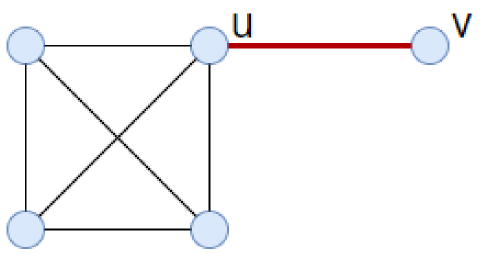
שיעור 7 - 30.11.2020

<u>רכיבי הדו-קשירות בגרף לא מכוון</u>

<u>הגדרות</u>

- **<u>רכיב דו-קשירות בגרף לא מכוון</u>** (הגדרה אינטואיטיבית) - רכיב שקשה להפריד בין קודקודיו.

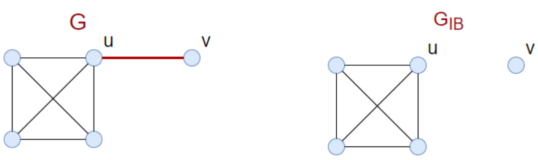
- **<u>רכיב קשירות (מתוך ויקיפדיה)</u>** – תת-גרף קשיר מקסימלי. כלומר, קבוצת קודקודים שיש מסלול בין כל שני קודקודים שלה והיא כוללת עם כל קודקוד גם את השכנים שלו. כל גרף מתפרק באופן יחיד לרכיבי קשירות.



- **<u>גשר</u>** - צלע $e = \{u, v\}$ בגרף לא מכוון $G = (V, E)$ תיקרא "גשר" ב-G אם הסרתה ממנו גורמת לכך שאין מסלול בין u ל-v.
  כלומר, הסרתה ממנו מגדילה באחד את מספר רכיבי הקשירות.
  בגרף משמאל, הצלע בין u ל-v היא גשר.

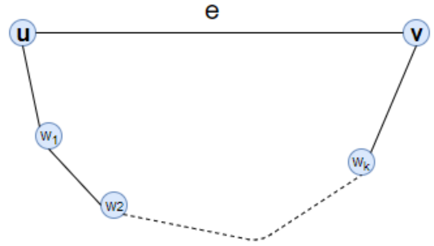
- **$\boldsymbol{G_{BI}}$** - בהינתן גרף $G = (V, E)$ נגדיר את הגרף $G_{BI} = (V, E_{BI})$ (BI עבור bi-connected) עם:
  $E_{BI} = \{e \in G \mid G - e \text{ אינה גשר ב } e\}$. כלומר, הסרנו מהגרף המקורי, G, את כל הגשרים.

- **<u>רכיב דו-קשירות</u>** - רכיבי הקשירות של G_{BI} נקראים רכיבי הדו-קשירות של G.
  למשל, אם נסיר את הגשרים מהגרף G שמצוייר בתחילת השיעור, נקבל גרף G_{BI} עם שני רכיבי קישור (אחד הוא v והשני הם כל שאר הקודקודים) – אלו רכיבי הדו-קשירות של G.



- **<u>קודקודים דו-קשורים</u>** – קודקודים u ו-v ייקראו דו-קשורים אם הם באותו רכיב דו-קשירות.

<u>טענה 1:</u>



יהי $G=(V,E)$ גרף לא מכוון.

צלע $e=\{u,v\}$ היא <u>אינה</u> גשר $\Leftrightarrow$ e חלק ממעגל ב-G.

<u>הוכחה:</u>

<u>צלע $e=\{u,v\}$ היא אינה גשר $\Rightarrow$ e חלק ממעגל ב-G:</u>

נניח כי e חלק ממעגל ב-G שנסמנו $u-v-w_1-\cdots-w_k-u$, אז הסרתה מ-G משאירה מסלול בין v ל-u: $v-w_1-\cdots-w_k-u$, לכן היא אינה גשר ב-G.

<u>צלע $e=\{u,v\}$ היא אינה גשר $\Leftarrow$ e חלק ממעגל ב-G:</u>

נניח ש-e היא אינה גשר ב-G. נסיר אותה, עדיין יש מסלול מ-u ל-v. נוסיף את e חזרה ל-G ונקבל כי היא סוגרת מעגל.

<u>טענה 2:</u>

ב-G_{BI} אין גשרים. זו נראית כמו טענה טריוויאלית, אבל לכאורה יכול להיות שתהליך הסרת הגשרים ב-G גרם ליצירה של גשר חדש ב-G_{BI}.

<u>הוכחה:</u>

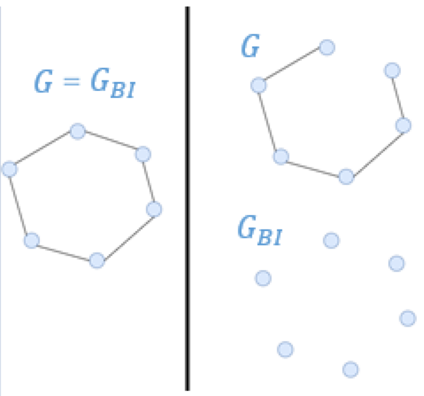
תהי $e\in E_{BI}$.

לפי האפיון בטענה הקודמת, e היא חלק ממעגל ב-G.

יתר הצלעות במעגל זה ב-G אינן גשרים ב-G (כי הן חלק ממעגל), ולכן כולן ב-G_{BI}.

לכן, e היא חלק ממעגל ב-G_{BI} $\Leftarrow$ e אינה גשר ב-G_{BI} (לפי טענה 1).

<u>הבחנות:</u>



- כשמסירים צלע מגרף, מס' רכיבי הקשירות יכול לגדול ב-1 לכל היותר.
- כשמסירים צלע מגרף, מס' רכיבי **הדו-קשירות** יכול לגדול ב- $|V|-1$ לכל היותר, כשהגרף הוא מעגל ומורידים צלע אחת.

  בחלק השמאלי של הציור, $G=G_{BI}$, כי אין גשרים – לכן יש רק רכיב דו-קשירות אחד. בחלק הימני הסרנו צלע מ-G, כעת כל הצלעות הן גשרים. G_{BI} הוא 6 קודקודים נפרדים ולכן יש 6 רכיבי דו-קשירות ב-G.

<u>משפט (ללא הוכחה)</u>

התנאים הבאים שקולים עבור גרף G:

1. u ו-v דו-קשורים ב-G.
2. בין u ו-v יש לפחות 2 מסלולים זרים בצלעות.
3. u ו-v חלק ממעגל ב-G.

<u>מציאת רכיבי הדו-קשירות בגרף</u>:

נבחין כי מציאת הגשרים ב-G מספיקה למציאת רכיבי הדו-קשירות בו, שכן אם נמצא אותם נוכל להסירם ולמצוא את רכיבי הקשירות ב-G_{BI} שהתקבל ב-$O(|V|+|E|)$.

<u>ניסיון 1 (נאיבי):</u>

לכל צלע $e=\{u,v\}\in G$ נסיר את e מ-G ונבדוק אם יש מסלול מ-u ל-v באמצעות DFS (או BFS, או אלגוריתם כלשהו לסריקת מסלולים בגרף).

<u>נכונות:</u> נובעת מהגדרת גשר.

<u>זמן ריצה</u>: לכל צלע העלות היא עלות DFS, כלומר, זמן הריצה הכולל הוא $O\big(|E|\cdot(|E|+|V|)\big)$.

<u>ניסיון 2</u>

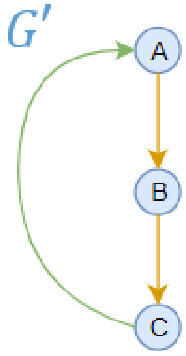
נרצה לאפיין את הגשרים באמצעות הרצה אחת של DFS.

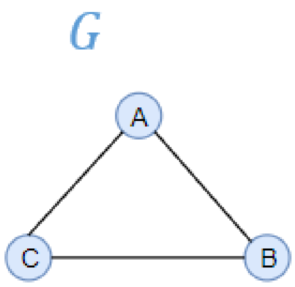
נזכיר כי בהרצת DFS על הגרף G מתקבל גרף מכוון G' שמתאים לסריקת הקודקודים ע"י האלגוריתם. G' מתקבל ע"י חיבור כל קודקוד v אליו הגענו במהלך הסריקה, אל הקודקוד u, ממנו הגענו, באמצעות צלע היוצאת מ-u ונכנסת ל-v.

ב-G', חלק מהצלעות מובילות אל קודקודים שטרם נסרקו – הן תקראנה "קשתות עץ" (הן נקראות כך כי אם נסיר את יתר הצלעות מ-G' נקבל יער, שהוא אוסף של עצים).

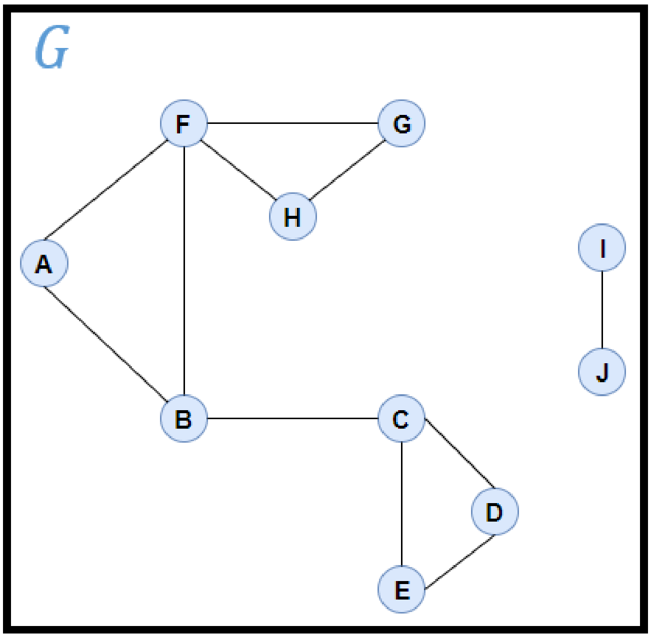
יתר הצלעות תקראנה "קשתות אחוריות".

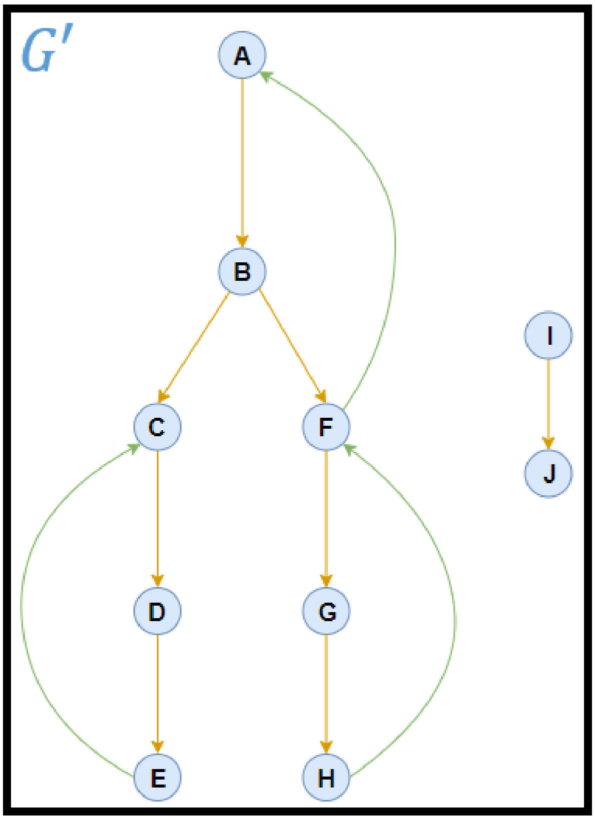
בגרף משמאל ניתן לראות את הגרף המקורי G ואת G', התוצר של תהליך ה-DFS, כאשר הצלעות הכתומות הן קשתות עץ והצלע הירוקה היא קשת אחורית.





נביט בדוגמה שבציור למטה - נתון הגרף G והרצנו עליו DFS בסדר הבא:
$A \to B \to C \to D \to E \to C$, לאחר מכן $B \to F \to G \to H \to F \to A$ ולסיום $I \to J$ וכך קיבלנו את G', בו קשתות העץ מסומנות בכתום והקשתות האחוריות בירוק.





<u>הבחנה:</u>

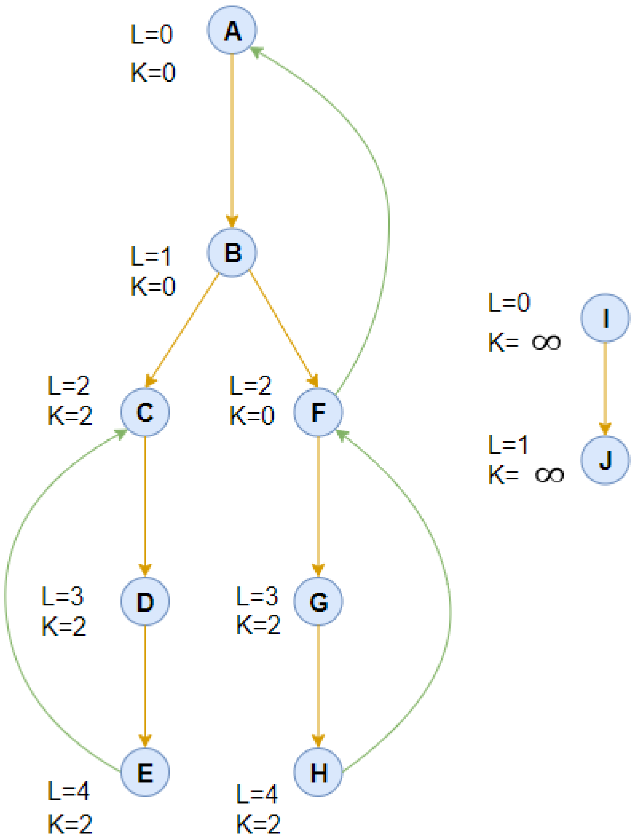
במציאת הגשרים ב-G אין צורך לנתח צלעות המהוות קשת אחורית בגרף DFS כלשהו של G.
הסבר: הן סוגרות מעגל ב-G', לכן הן גם חלק ממעגל ב-G (כי לכל צלע ב-G' יש גרסה לא מכוונת ב-G) ולכן הן בוודאות לא גשרים ב-G.

נגדיר לכל קודקוד v ב-G' את הערך $\mathrm{K(v)}$ המוגדר באופן הבא:

$$\mathrm{K(v)} = \min\left\{ \mathrm{L(w')} \;\middle|\; \begin{array}{c} \text{w צאצא של v בעץ} \\ \text{ו} - \mathrm{(w,w')} \text{ קשת אחורית} \end{array} \right\}$$

כלומר, מסתכלים על כל הצאצאים של v בעץ v (כולל v עצמו), מסתכלים על כל הקשתות האחוריות (w, w') שיוצאות מהם ולוקחים את ה-w' הגבוה ביותר בעץ (כלומר עם הרמה הנמוכה ביותר).

נביט בערכי K ו-L (הרמה) של כל הקודקודים ב-G' מהדוגמה:

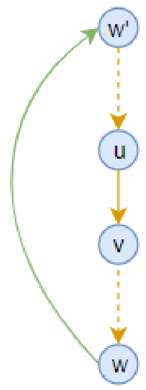


<u>טענה 3</u>:

יהי $G = (V, E)$ גרף לא מכוון ויהי G' גרף DFS המתאים לו.

תהי $e = (u, v)$ (צלע מכוונת) קשת עץ ב-G'.

אזי e **אינה** גשר ב-G $\Leftrightarrow$ $K(v) < L(v)$.

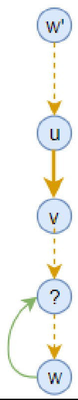


<u>הוכחה:</u>

<u>$\Rightarrow$:</u>

נניח כי $K(v) < L(v)$.

אזי $K(v) \leq L(u)$ (כי u הוא אביו של v) ולכן e חלק ממעגל ב-G.



<u>$\Leftarrow$:</u>

נניח כי $K(v) \geq L(v)$. אזי כל הצאצים של v מובילים בקשת אחורית אל קודקודים בתת-עץ ש-v שורשו ובפרט נמוכים מ-u, לכן $e = \{u, v\}$ לא סוגרת מעגל.

<u>הבחנה</u>:

$$K(v) = \min\left(\left\{L(w')\middle|\begin{matrix}\text{קשת אחורית } (v,w')\\ G' - \text{ב}\end{matrix}\right\} \cup \left\{K(v')\middle|\begin{matrix}\text{קשת } (v,v')\\ G' - \text{עץ ב}\end{matrix}\right\}\right)$$

בעזרת ההבחנה נוכל לקבל את כל ערכי L ו-K של הקודקודים בעזרת הרצה אחת של DFS על G.

לכל קודקוד נבדוק את קיום טענה 3 ונסמן צלע שנכנסת אליו כגשר במקרה הצורך.

<u>הערה טכנית לגבי מציאת ערכי L ו-K באמצעות ה-DFS:</u>

כדי למצוא את הרמות, בירידה מקודקוד אחד לבנו מעבירים לבן, בקריאה הרקורסיבית, את רמת האב וכך הבן יודע להוסיף לה 1. לעומת זאת, את K מקבלים מפעפוע מעלה בקריאה הרקורסיבית של האלגוריתם: כשנחשב את ה-K של קודקוד v אנחנו רוצים לדעת את הרמה של כל הבנים שלו וגם את ערכי ה-K של הבנים שלו - את שניהם נקבל אחרי שקראנו רקורסיבית ל-DFS על הבנים.

<u>נכונות</u>: נובעת מטענה 3.

<u>זמן ריצה</u>: הרצה אחת של DFS ומילוי הערכים בהתאם בעלות הרצת DFS - $O(|V| + |E|)$.

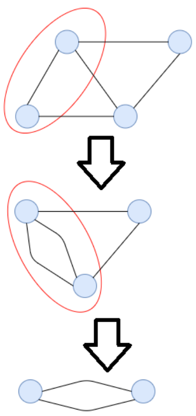
הערה: באופן דומה, ניתן להגדיר רכיבי דו-קשירות לפי קודקודים מפרידים במקום צלעות מפרידות.

<u>מציאת חתך מינימלי (מבחינת מספר הצלעות) בגרף</u>

<u>הגדרות:</u>

- **<u>מולטי-גרף</u>** - מוגדר בדומה לגרף, למעט כך שבין זוג קודקודים תיתכן יותר מצלע אחת.
- **<u>חתך בגרף</u>** – קבוצת צלעות שהסרתן מהגרף מגדילה ב-1 את מספר רכיבי הקשירות בו.

נרצה לתאר אלגוריתם שבהינתן מולטי-גרף G מחזיר את גודל החתך המינימלי בו בהסתברות מספיק טובה.



<u>האלגוריתם:</u>

1. בצע עד להיוותרות שני קודקודים בגרף G:
   א. הגרל צלע ב-G באופן מקרי (אחיד).
   ב. חבר את שני הקודקודים לקודקוד יחיד (כל צלעותיו הן אלו שיצאו מאחד מהם, למעט בין שניהם).
2. החזר את מספר הצלעות ביניהם.