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 $i \le j$.

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

2	1	5	6

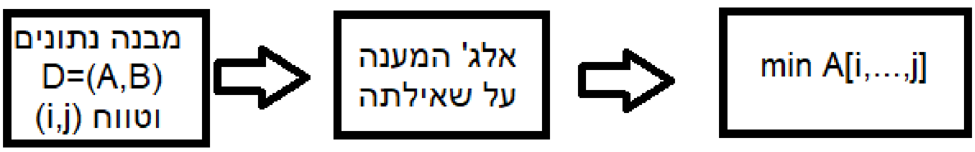
מדובר בבעיה שמכילה שאילתות, לכן נרצה לבצע עיבוד מקדים כלשהו למערך, כלומר, לחלץ ממנו מידע מסוים ואז להשתמש במידע זה כדי לענות על כל שאילתה שנקבל בצורה יעילה.

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

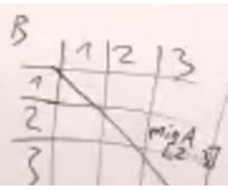
RMQ1 – ללא עיבוד מוקדם:

- **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:

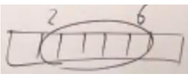
שם הפתרון	T_p	T_q	S
RMQ1	$O(1)$	$O(n)$	$O(n)$

(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 $\mathrm{B[i,j]} = min\ \mathrm{A[i,\dots,j]}$ 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 $\mathrm{A}[2,5]$.
2. $\mathrm{A}[6]$.

משווים בין שתי האפשרויות ומחזירים כתשובה את המינימלית מבניהן.

נמלא את הטבלה באופן הבא:

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

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

- 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 תופסת 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 המינ' עבורם בעזרת המערך B.
2. נמצא מינימום ב"זנבות" (קצוות הטווח שאולי לא מהווים בלוק שלם – ראו דוגמה).
3. נחזיר את האיבר המינימלי שהתקבל עבור קבוצת ערכי המינימום של הבלוקים והזנבות.

<u>נביט בדוגמה:</u>

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

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

- 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 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:

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

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 –**

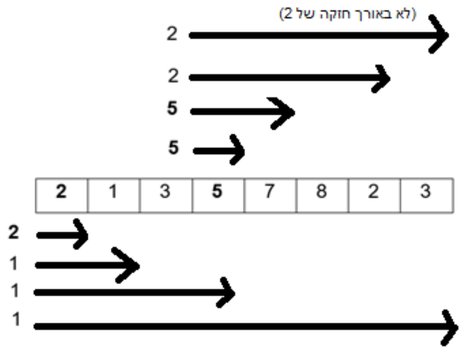
1. נגדיר טבלת B עם

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

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.)



דוגמה: 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 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

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

*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 ($A[i]$) לא חורגים מסופו ($A[j]$).
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>

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

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

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

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

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 השמאלי של התת-קטע השני. פורמלית, נרצה להראות כי $\boldsymbol{i + 2^{\lfloor \log(t) \rfloor} - 1 \ge j - 2^{\lfloor \log(t) \rfloor} + 1}$:

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

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	$O(n \cdot log(n))$	$O(1)$	$O(n \cdot log(n))$

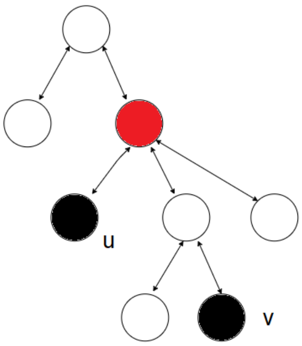
בעיית ה-LCA (אב קדמון משותף מינימלי - Lowest Common Ancestor)

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.

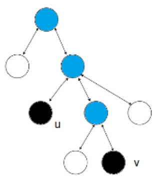
הגדרת הבעיה: בהינתן עץ מושרש (מדובר בעץ כללי – לא חיפוש ולא בינארי!), מעוניינים לענות על שאילתות מהצורה "מהו הקודקוד הנמוך ביותר ש-u ו-v צאצאיו?", בהינתן u ו-v בשאילתה.

(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.

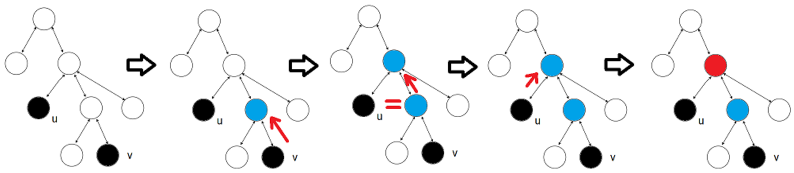
ניתוח סיבוכיות האלגוריתם:

- T_p - $\boldsymbol{O(n)}$, נקצה מערך בגודל $O(n)$ עבור תיעוד הקודקודים שפגשנו בדרך לשורש מ-v.
- 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 - $(O(n), O(n), O(n))$:

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

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

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.

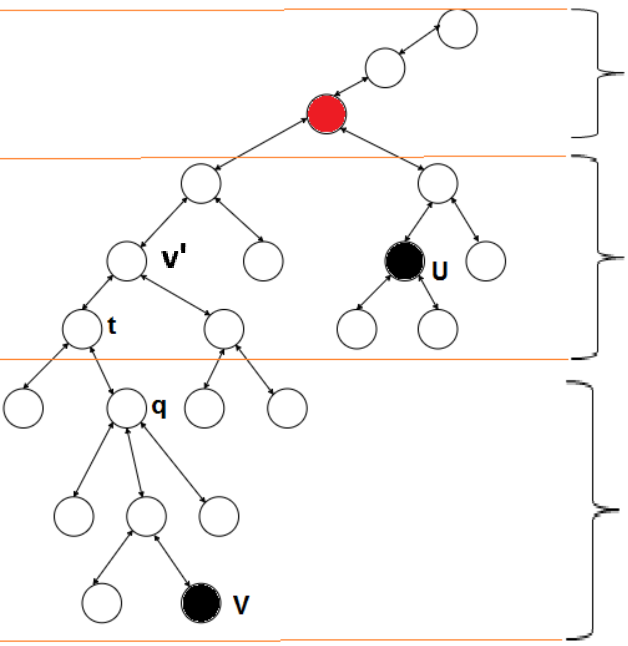
לכן, נחלק את גובה העץ לבלוקים המכילים $\sqrt{h}$ רמות כל אחד, כך שכל בלוק מכיל אמנם מספר שונה של קודקודים אך מספר זהה של רמות.

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:

$GP[v]\begin{cases} v.parent \\ GP[v.parent] \end{cases}$	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 זה הגענו לקודקוד שהוא הנמוך בקבוצתו וכן אב קדמון משותף, אך 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.



__דוגמה:__

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 $\boldsymbol{O(\sqrt{n})} \le \boldsymbol{O(\sqrt{h})}$.
- S – $\boldsymbol{O(n)}$, העץ עצמו + גודל GP, שגודלה $O(n)$.

LCA4 – חישוב אב קדמון במרחק 2^k לכל קודקוד ולכל k –

:$\left(O(n\cdot log(n)), O(log(n)), O(n\cdot log(n))\right)$

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-$B[v,k]$ יכיל את האב הקדמון של v במרחק 2^k ממנו.

  נמלא את הטבלה באמצעות תכנון דינאמי באופן הבא:

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

כלומר, עבור $k\neq 0$ 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 $B[v,k-1]$ ואז מהקודקוד אליו הגענו קופצים שוב 2^{k-1} באמצעות הסתכלות על $B[B[v,k-1],k-1]$.

- **מענה על שאילתה:** 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 $B[v,k]$ אינו גבוה מ-u עדכן $v\leftarrow B[v,k]$.

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

- T_p - $\boldsymbol{O(n\cdot log(n))}$, 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 $O(n\cdot\log(n))$.
- T_q – $\boldsymbol{O(log(n))}$, יש לנו $O(\log(n))$ איטרציות, כל איטרציה עולה $O(1)$.
- S – $\boldsymbol{O(n\cdot log(n))}$, כפי שמתואר בניתוח 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.

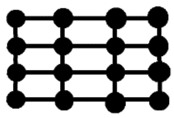
2. שילוב אלגוריתמים

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.

אלגוריתם 1:

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 $O\big(n\cdot log(n)\big)$ – בגלל שהמיון עולה $\boldsymbol{O\big(n\cdot log(n)\big)}$.

אלגוריתם 2:

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

זמן ריצה של אלגוריתם 1:

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.

זמן הריצה של אלגוריתם 2:

שוב נסמן ב-m את כמות סוגי האותיות.

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

אז זמן הריצה בנוי ממיון ומזמן הרצת BFS עבור כל סוג של אות (אולי יש כמה הרצות במקביל לכל אות, אבל זה לא מייקר 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:

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

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

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

- 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):

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

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

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

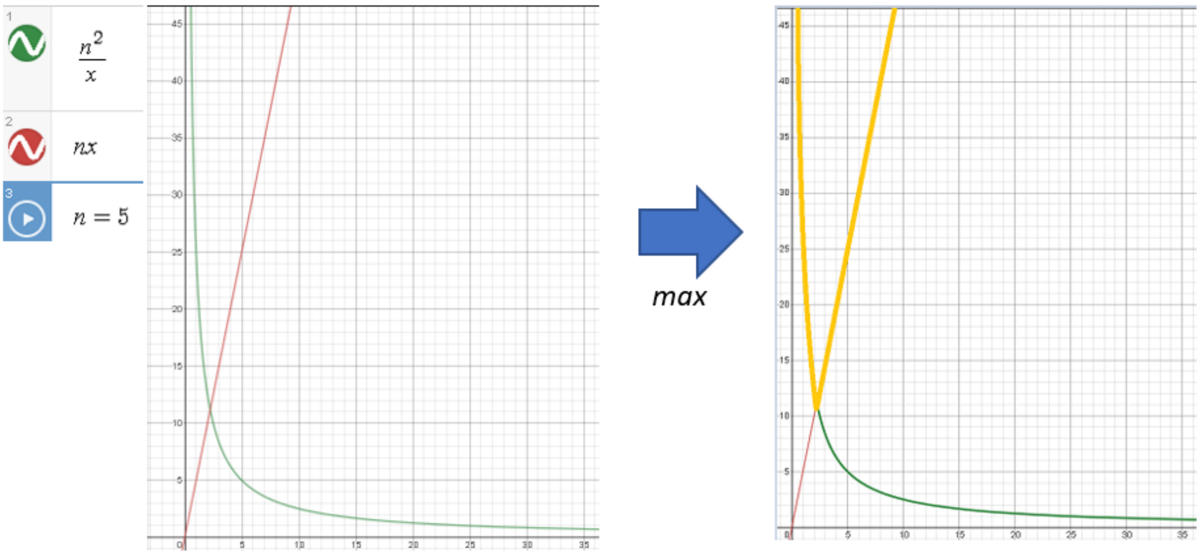
וזה זמן יותר טוב מ-$O(n^2)$, שזה זמן הריצה של שני האלגוריתמים הקודמים.

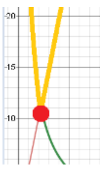
ii. <u>דרך שהיא יותר אינטואיציה גיאומטרית יפה (לא תמיד תסתדר יפה עם פו' אחרות):</u>

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:

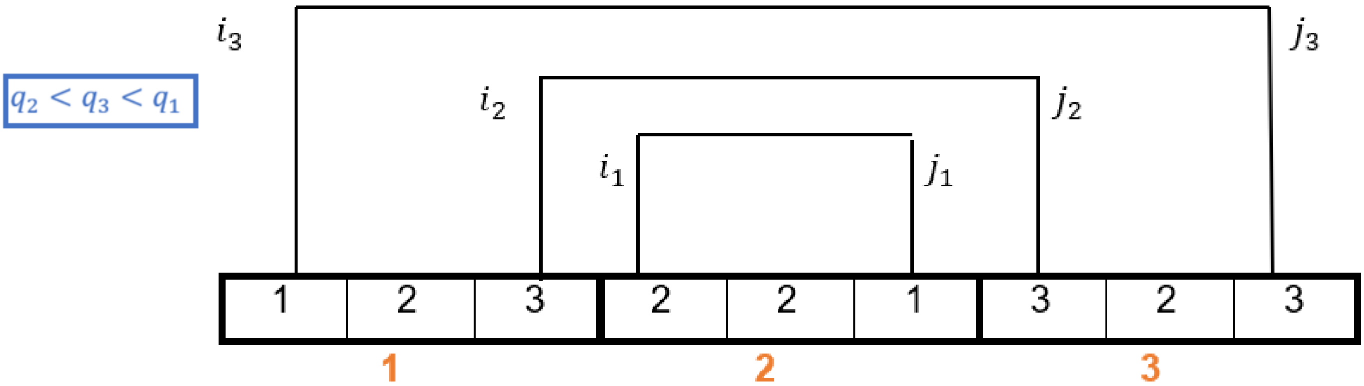
$$\boldsymbol{\frac{n^2}{k}=k\cdot n\Leftrightarrow n=k^2\Leftrightarrow \boxed{k=\sqrt{n}}}$$

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

3. אלגוריתם Mo

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 המבקשות 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] $\left\lfloor \frac{i_1}{\sqrt{n}} \right\rfloor < \left\lfloor \frac{i_2}{\sqrt{n}} \right\rfloor$ (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 $\left\lfloor \frac{i_1}{\sqrt{n}} \right\rfloor = \left\lfloor \frac{i_2}{\sqrt{n}} \right\rfloor$ 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 בו 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.

<u>דוגמה:</u>

ספירת מס' האיברים השונים בטווח.

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

<u>נריץ את האלגוריתם על הדוגמה:</u>

- 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 באחד את המונה מהשאילתה הקודמת/ המונה שאתחלנו.

<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 -** $\mathbf{O\left(n + m \cdot \sqrt{n}\right)}$:
   - 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 $\mathrm{O}\left(\sqrt{\mathrm{n}}\right)$ צעדים (כי גודל הבלוק הוא $\sqrt{n}$), אז את כל התזוזות של L בתוך בלוקים ניתן לחסום ע"י $\mathrm{O}\left(\mathrm{m} \cdot \sqrt{\mathrm{n}}\right)$ (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: $\mathbf{O}\left(\boldsymbol{n} + \mathbf{m} \cdot \sqrt{\mathbf{n}}\right)$.

<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-$O\left(n\sqrt{n}\right)$.

<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? (העשרה)

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 $O(n \cdot \sqrt{n})$, 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-$O\left((m+n)\cdot\sqrt{n}\right) = O(n\sqrt{n})$. 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 $\boldsymbol{\Omega(n\sqrt{n})}$ (כלומר, לפחות $n\cdot\sqrt{n}$).

נביט בדוגמה הבאה:

השאילתה הראשונה הין בין הטווח $[0,\sqrt{n}]$, השאילתה השנייה היא בין $[0,2\cdot\sqrt{n}]$ וכן הלאה עד לטווח $[0,(\sqrt{n}-1)\sqrt{n}]$. השאילתה הבאה היא בין הטווח $[\sqrt{n},2\cdot\sqrt{n}]$, זו שאחריה בטווח $[\sqrt{n},3\cdot\sqrt{n}]$ 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:

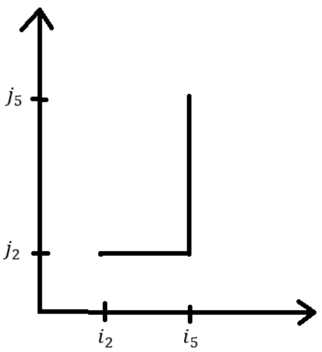
1. מזיז את L לפחות $\sqrt{n}$ צעדים ימינה.

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 (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 $a_1, \dots, a_n$ 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.

<u>הוכחה:</u>

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

<u>הערות:</u>

- <u>במעבר הראשון</u> אנחנו אומרים שסכום כל איברי הסדרה ≤ סכום t האיברים הראשונים שלה ($t \leq n$).
- <u>במעבר השני</u> אנחנו מניחים, בלי הגבלת הכלליות, ש-t האיברים הייחודיים בתחילת הסדרה.
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.
- <u>במעבר השלישי</u> 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.

<u>ניסיון 1 (נאיבי, לא משתמש בערכים ייחודיים):</u>

הגדר נוסחה שתתאים לטבלת התכנון הדינאמי:

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

נגדיר את הטבלה כך:

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

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.

<u>זמן הריצה של ניסיון 1:</u>

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:

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

<u>ניסיון 2:</u>

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 =	1	2	1
A =	1	2	7

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.

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

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

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

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 $n \cdot K \geq n \cdot log(n)$ 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 - $\left(O(n), O(1), O(n)\right)$:

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

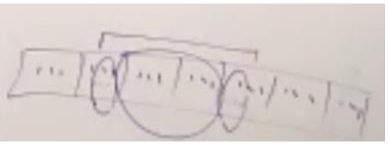
ניסיון 1:

- **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 הזנבות עם RMQ4. **עלות:** $\boldsymbol{O(1)}$
3. החזר את האיבר המינימלי מבין שלושת האיברים שהתקבלו. **עלות:** $\boldsymbol{O(1)}$



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

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

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:

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

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:

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

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

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 |

3	4	3

0	1	0

הבחנות:

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

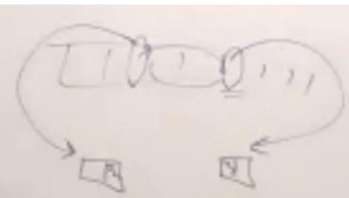
- **עיבוד מוקדם:**

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 - $\boldsymbol{(O(n),O(1),O(n))}$:</u>

- **מענה על שאילתה -** $\boldsymbol{O(1)}$

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:

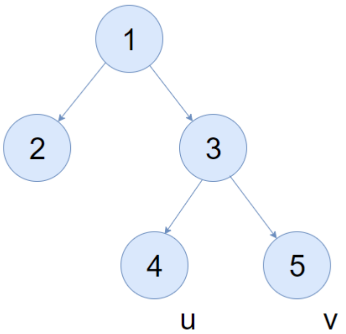
$$O(2^{L-1}\cdot L\cdot \log(L)) \le O(2^{L}\cdot L\cdot \log(L)) = O\left(2^{\frac{1}{2}\log(\mathrm{n})}\cdot L\cdot \log(L)\right)$$

$$= O\big(\sqrt{n}\cdot \log(n)\cdot \log(\log(n))\big) \le \boldsymbol{O(n)}$$

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

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

:LCA5



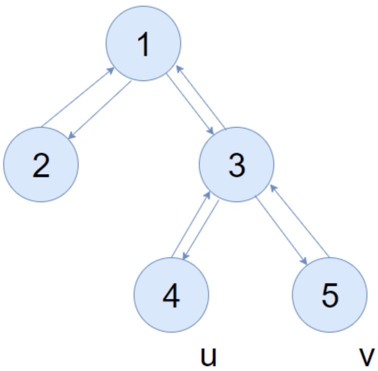
נרצה לתרגם את העץ למערך ע"י סריקתו. נרצה שהסריקה תקיים מספר תכונות:

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:

				v		u		
1	2	1	3	4	3	5	3	1

$E^1 =$ [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:

$$E^1\left[\operatorname{argminRMQ^*}_{E^2}\left[\bar{E}[v],\bar{E}[u]\right]\right]$$

- $\bar{E}[v]$ – האינדקס של v ב-E^1 ו-E^2.
- $\bar{E}[u]$ – האינדקס של u ב-E^1 ו-E^2.
- $\operatorname{argminRMQ^*}_{E_2}\left[\bar{E}[v],\bar{E}[u]\right]$ – מחפשים את הקודקוד עם הדרגה המינימלית בקטע שבין v ל-u ב-E^1, לכן מחפשים מינימום על הקטע $\left[\bar{E}[v],\bar{E}[u]\right]$ 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.
- $E^1\left[\operatorname{argminRMQ^*}_{E_2}\left[\bar{E}[v],\bar{E}[u]\right]\right]$ – הקודקוד עם הדרגה המינימלית בקטע בין v ל-u ב-E^1.

<u>ניתוח זמן ריצה:</u>

- **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: $O(1)$.**

[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.

__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.

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.

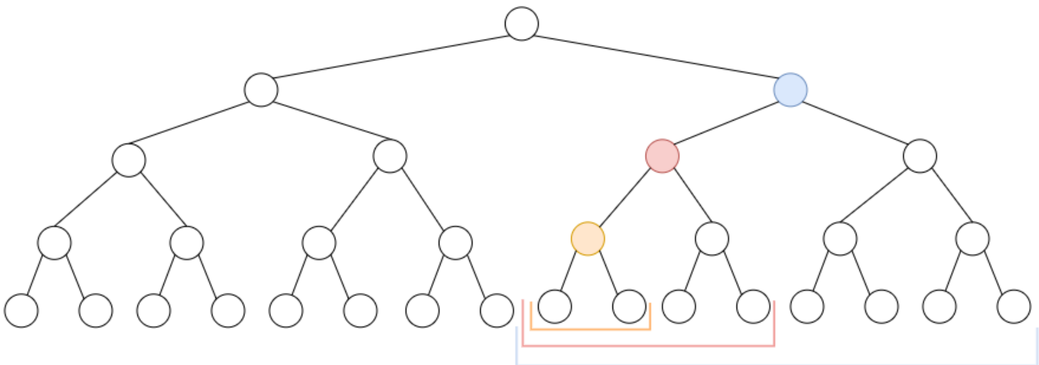
דרישות: פעולות השאילתה והעדכון בעלות $\boxed{O(log(n))}$.

מבנה הנתונים:

בהינתן סדרה $a_1, \dots, a_n$ נגדיר 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 מלא! 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.

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

***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.

<u>דוגמאות לאופרטורים חוקיים:</u>

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

<u>**מימוש הפעולות – כדי להקל על הכתיבה עוברים לעבוד על מקרה פרטי בו f היא סכום:**</u>

<u>אתחול (A):</u>

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 $O(n)$.

<u>עדכון (i,x):</u>

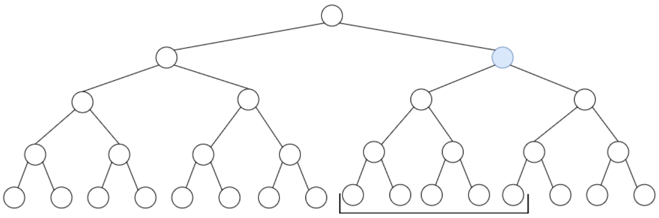
הגדר את P[i] (כלומר, את מי ש-P[i] מצביע עליו) להיות x ולכל אב קדמון שלו (החל מהעלה כלפי מעלה), v, עדכן: $\boldsymbol{v.value=v.left.value+v.right.value}$.

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 $[i, j]$.

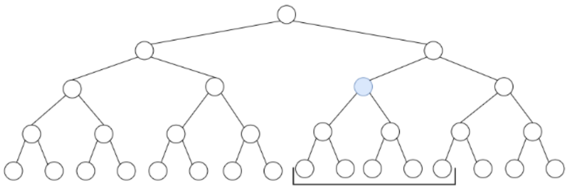
למשל, בדוגמא למטה, הקודקוד התכלת הוא קודקוד הפיצול עבור שאילתה על הקטע המסומן:



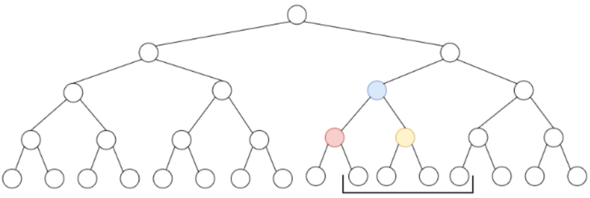
1. התקדם מהשורש עד למציאת **קודקוד הפיצול**, v (באמצעות בדיקת i ו-j למול ערכי a ו-b).
2. חשב את ערך (סכום במקרה שלנו) **הסיפא** של $v.left$ שמתחילה ב-i באופן הבא:
   - כל עוד $v \neq null$:

     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.

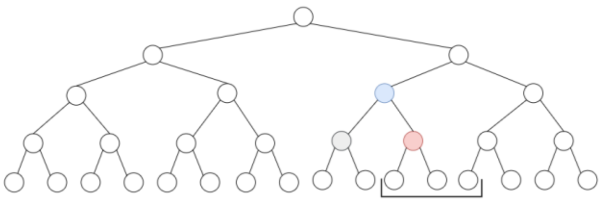


     ב. אם i מוכל בקטע השליטה של v.left, הוסף את v.right.val לסכום ועדכן את v להיות v.left. למשל, בדוגמה משמאל, כשנגיע במהלך חישוב הסיפא לקודקוד הכחול נוסיף את ה-value של הקודקוד הצהוב לסכום ונמשיך לורוד.



     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 $i \le t \le j$ עדכן את a_t להיות $x \circ a_t$

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

<u>תכונת עץ הקטעים הדואלי:</u>

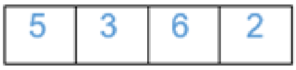
ערכו של כל איבר a_i מתקבל ע"י חישוב:

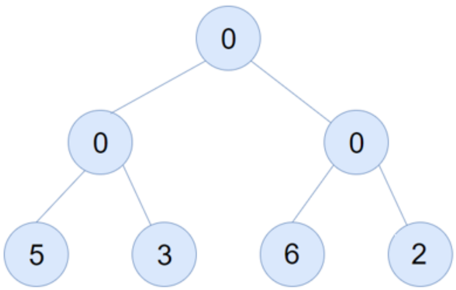
$$v_k.val \circ v_{k-1}.val \circ \dots \circ v_1.val$$

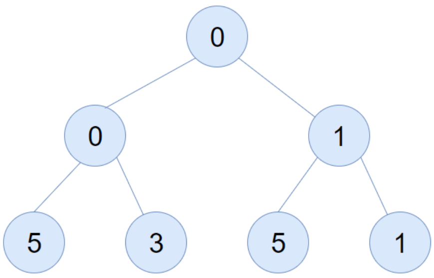
כאשר v_1 הוא העלה המתאים ל-a_i (שזה אותו הדבר כמו P[i]) ו-v_k הוא שורש העץ,

כלומר, $v_k, v_{k-1}, \dots, v_1$ 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>

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

מימוש הפעולות:

1. אתחול(A):

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. **שאילתה (i):**

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

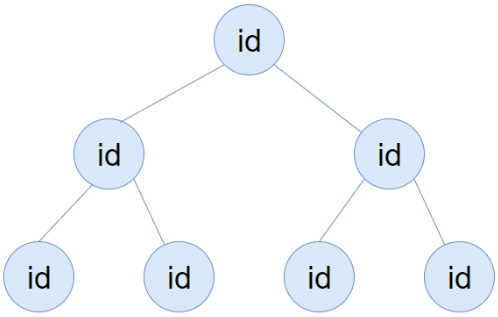
3. **עדכון (i,j,x):**

ניסיון 1:

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:

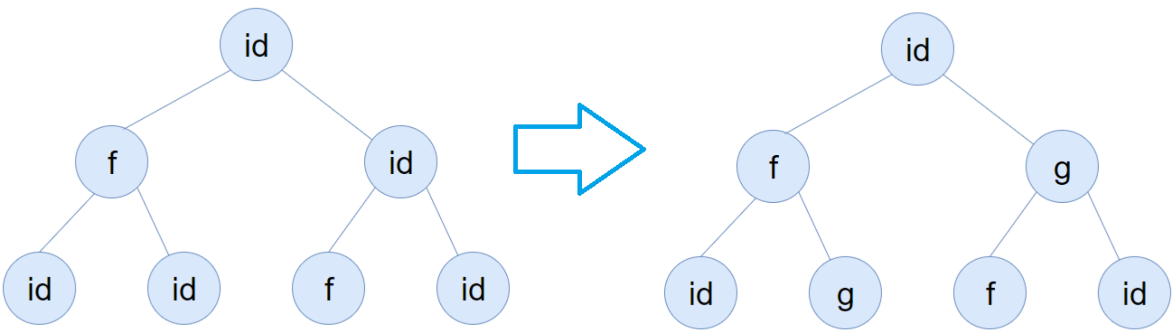


$(1,3,f)$

$(2,4,g)$

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

נפעיל את f על הקודקודים הרלוונטיים ואז את g:

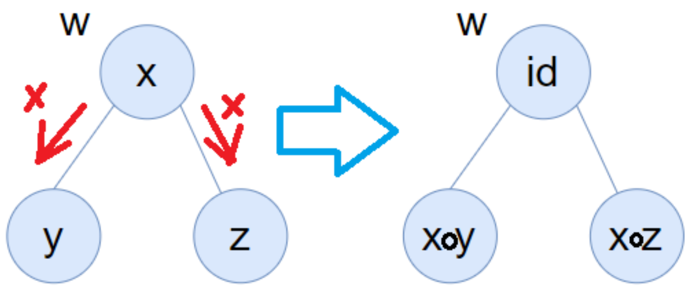


אוי לא! זה לא עבד! אמנם עבור העלה השלישי משמאל נקבל $g \circ f$ אם נעלה ממנו אל השורש ונרכיב משמאל כל פונקציה שנפגוש, אבל אם נפעל באותה דרך עבור העלה השני נקבל $f \circ g$.

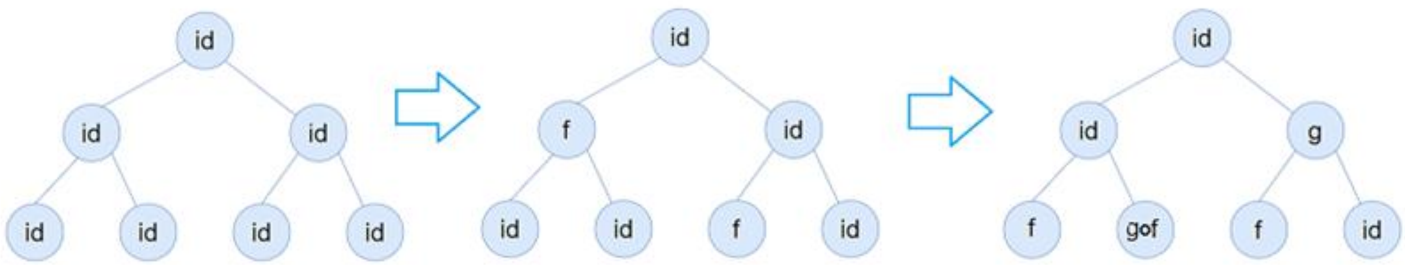
הערה: גם אם היינו מנסים לשנות רק את הדרך בה אנחנו קוראים את העץ – למשל מלמעלה למטה בתת-עץ השמאלי וההפך בימני זה עדיין לא היה עובד כי אפשר היה לקבוע שתי סדרות שונות של פעולות עדכון שמניבות את אותו העץ ואז לא נדע לפי איזו סדרה לקרוא אותו.

ניסיון 2 (תומך באופרטורים לא קומוטטיביים):

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 $p.val$.

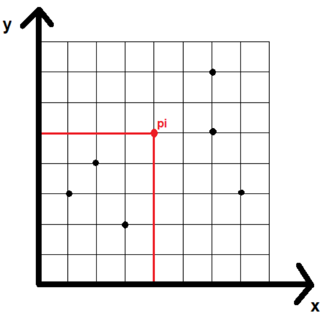
הנקודות נסרקות אחת אחרי השניה לפי הסדר בו נתונות בקלט. בתהליך זה, באיטרציה ה־i מסומנת הנקודה p_i אלא אם סומנה כבר קודם, ובהמשך מסומנת באותה איטרציה כל נקודה p_j שעדיין לא סומנה, ואשר עבורה קיימות לפחות $p_j.val$ נקודות **שכבר סומנו** (יתכן באותה איטרציה) שקואורדינטת ה־x שלהן קטנה מ־$p_j.x$ וקואורדינטת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

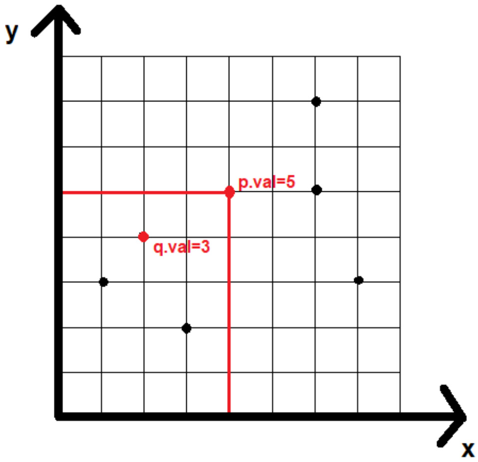
אתחלנו את ערכה של נקודה p_j במערך הפלט להיות j.

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.

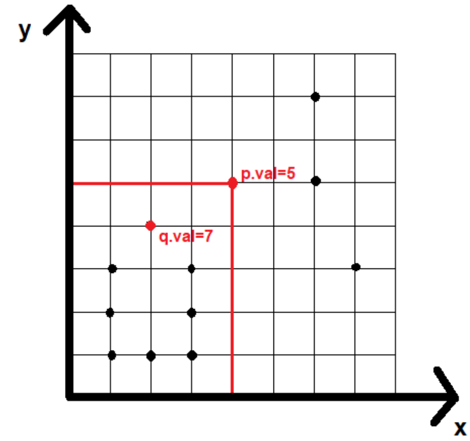
__פתרון שני בעלות קטנה מ-$O(n^2)$:__

__הבחנות__

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

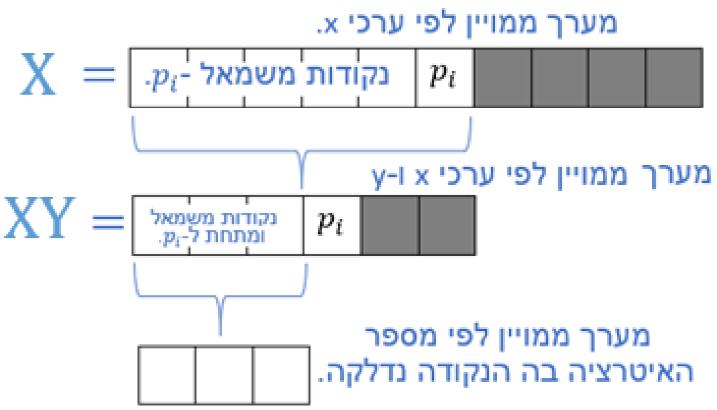


- אם $q.val < p.val$ אז, אינדוקטיבית, זמן הסימון של q נכון.



- 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.



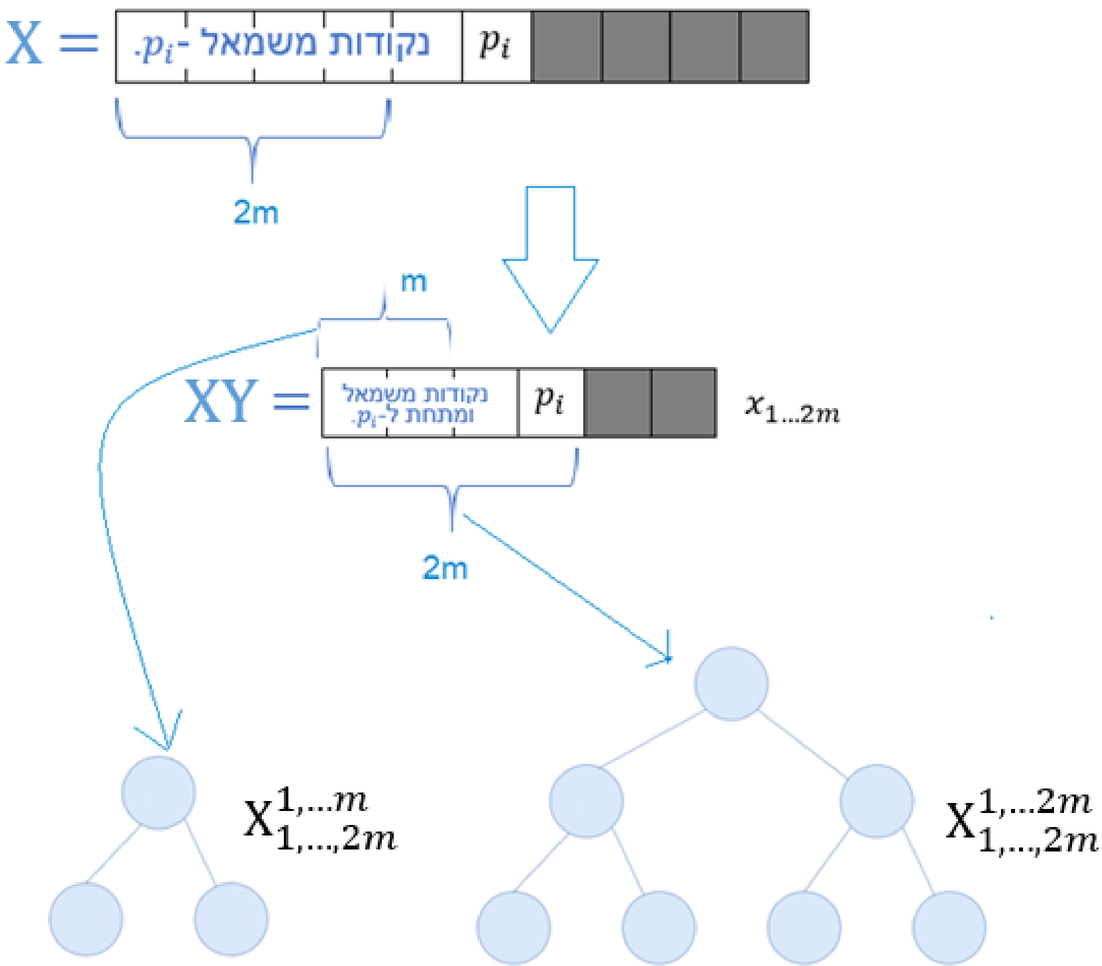
[8] "רישא" הוא החלק של המערך שמשמאל לנקודה p_i שנסרקת כרגע, באיטרציה ה-i.

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?".

<u>עיבוד מוקדם:</u>

1. לכל נקודה p, עדכן את איטרציית הסימון הדיפולטיבית שלה.
2. הגדר מערך X של הנקודות ממוינות לפי ערכי x וכן מיפוי הפוך $\bar{\mathrm{X}}$.
3. נמיין לפי y את כל הרישות של X באורכים $m, 2m$ וכן הלאה, ונשמור את המערכים במערכים שנקרא להם $\mathrm{X}_{1\ldots m}, \mathrm{X}_{1\ldots 2m}$ 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 $\mathrm{X}_{1,\ldots,k\cdot m}^{1,\ldots m}, \mathrm{X}_{1,\ldots,k\cdot m}^{1,\ldots 2m}, \ldots$ 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:

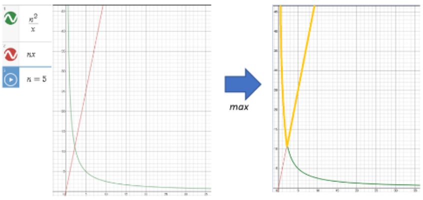
1. מצא את $V[i]$ ב-X בעזרת $\bar{X}$ ואת הרישא המקסימלית הממויינת לפי y ומתאימה לה, $X_{1\dots km}$.
2. מצא את המיקום המתאים ל-$V[i]$ ב-$X_{1\dots km}$ ואת עץ הרישא המקסימלית המתאימה לה שממוין לפי זמן סימון, $X_{1,\dots,k\cdot m}^{1,\dots t\cdot m}$.
3. הוסף נקודות מהזנבות בהתאם לצורך (= אם יש זנבות ואם יש בהם איברים עם ערך y קטן מהמקס' של העץ. למשל, בציור לעיל יש איבר מפוספס בין p_i לסוף הרישא $x_{1\dots 2m}$ – 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 $X_{1,\dots,km}$ - $O\left(\frac{n}{m}\cdot n\cdot\log(n)\right)$.
  4. מיון העצים $X_{1,\dots,k\cdot m}^{1,\dots t\cdot m}$ - $O\left(\frac{n^2}{m^2}\cdot n\cdot\log(n)\right)$.
  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:

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

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

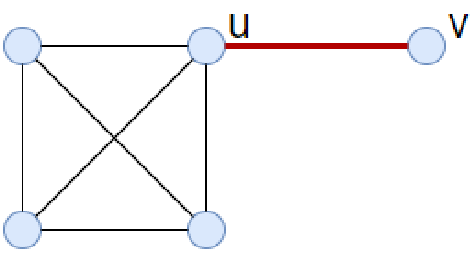
שיעור 7 - 30.11.2020

רכיבי הדו-קשירות בגרף לא מכוון

הגדרות

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

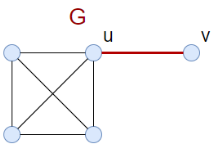
- **רכיב קשירות (מתוך ויקיפדיה)** – תת-גרף קשיר מקסימלי. כלומר, קבוצת קודקודים שיש מסלול בין כל שני קודקודים שלה והיא כוללת עם כל קודקוד גם את השכנים שלו. כל גרף מתפרק באופן יחיד לרכיבי קשירות.

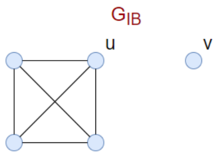


- **גשר** - צלע $e = \{u, v\}$ בגרף לא מכוון $G = (V, E)$ תיקרא "גשר" ב-G
אם הסרתה ממנו גורמת לכך שאין מסלול בין u ל-v.
כלומר, הסרתה ממנו מגדילה באחד את מספר רכיבי הקשירות.
בגרף משמאל, הצלע בין u ל-v היא גשר.

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

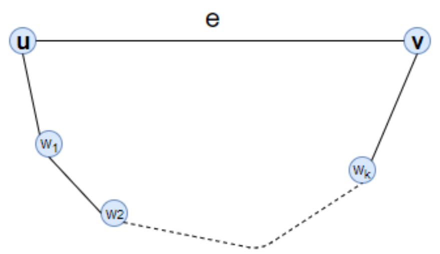
- **רכיב דו-קשירות** - רכיבי הקשירות של G_{BI} נקראים רכיבי הדו-קשירות של G.
למשל, אם נסיר את הגשרים מהגרף G שמצוייר בתחילת השיעור, נקבל גרף G_{BI} עם שני רכיבי קישור (אחד הוא v והשני הם כל שאר הקודקודים) – אלו רכיבי הדו-קשירות של G.





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

טענה 1:



יהי $G = (V, E)$ גרף לא מכוון.

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

הוכחה:

<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.

טענה 2:

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}.

הוכחה:

תהי $e \in E_{BI}$.

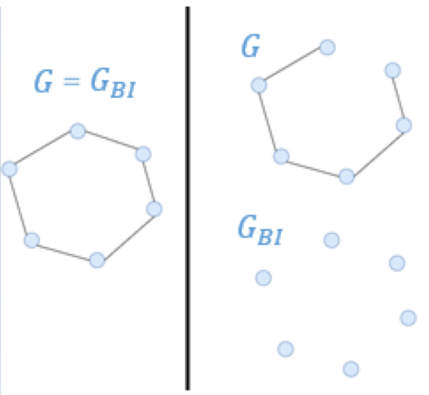
לפי האפיון בטענה הקודמת, e היא חלק ממעגל ב-G.

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).

הבחנות:

- 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.

משפט (ללא הוכחה)

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.

מציאת רכיבי הדו-קשירות בגרף:

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-$O(|V|+|E|)$.

ניסיון 1 (נאיבי):

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).

נכונות: נובעת מהגדרת גשר.

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

ניסיון 2

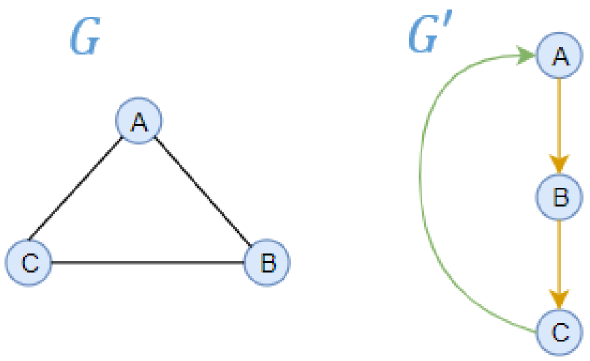
נרצה לאפיין את הגשרים באמצעות הרצה אחת של DFS.

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.

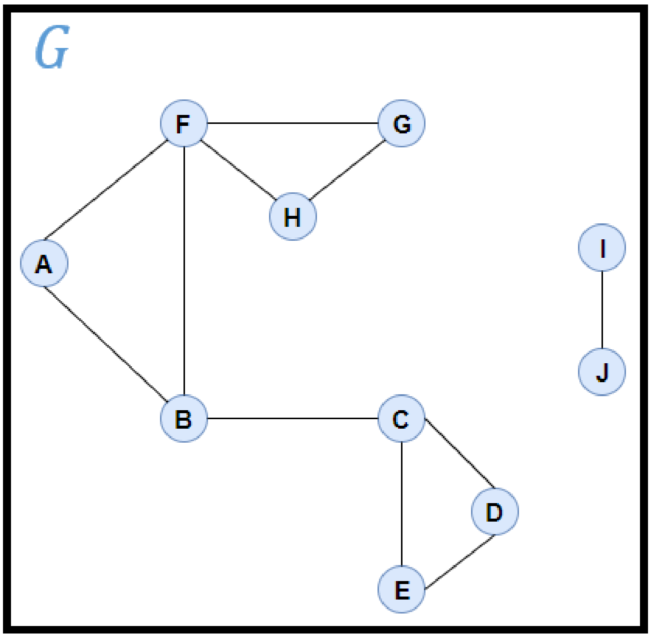
ב-G', חלק מהצלעות מובילות אל קודקודים שטרם נסרקו – הן תקראנה "קשתות עץ" (הן נקראות כך כי אם נסיר את יתר הצלעות מ-G' נקבל יער, שהוא אוסף של עצים).

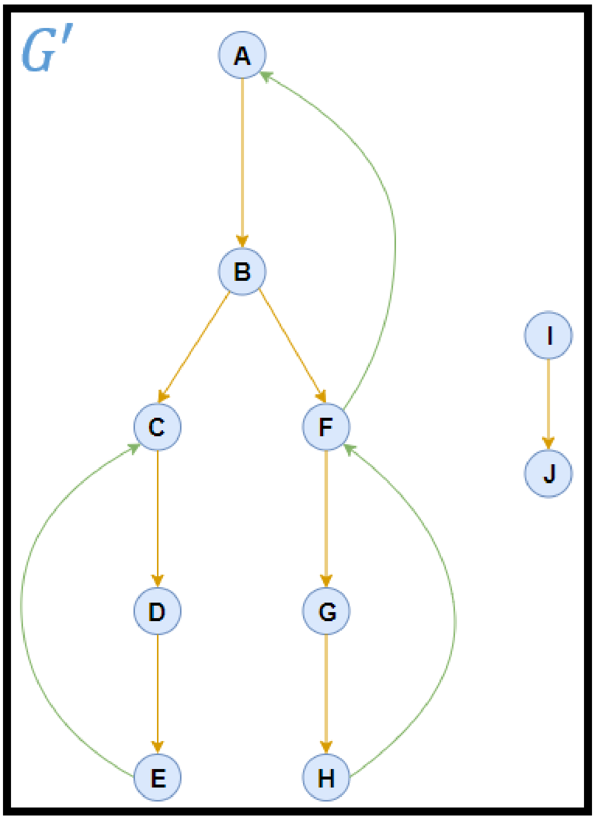
יתר הצלעות תקראנה "קשתות אחוריות".

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>

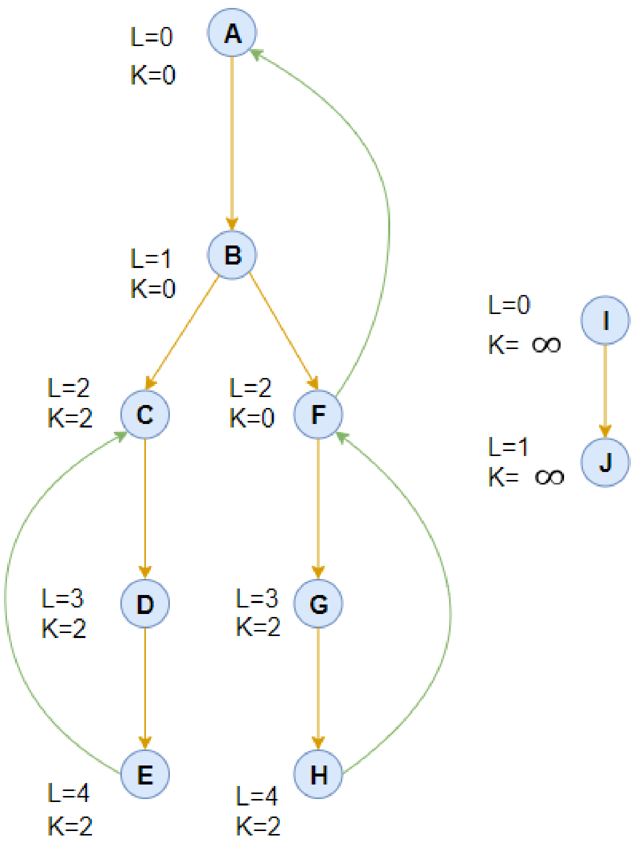
במציאת הגשרים ב-G אין צורך לנתח צלעות המהוות קשת אחורית בגרף DFS כלשהו של G.
הסבר: הן סוגרות מעגל ב-G', לכן הן גם חלק ממעגל ב-G (כי לכל צלע ב-G' יש גרסה לא מכוונת ב-G) ולכן הן בוודאות לא גשרים ב-G.

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 (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:

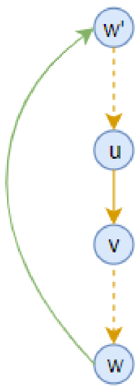


<u>טענה 3</u>:

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 $\Leftrightarrow$ $K(v) < L(v)$.

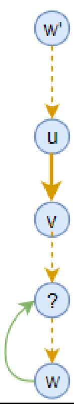


<u>הוכחה:</u>

<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>:

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

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 - $O(|V|+|E|)$.

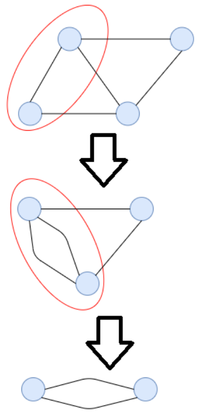
הערה: באופן דומה, ניתן להגדיר רכיבי דו-קשירות לפי קודקודים מפרידים במקום צלעות מפרידות.

<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.

שיעור 8 – 7.12.2020

מציאת חתך מינימום בגרף - המשך

הבהרה:

אנחנו מדברים פה על חתך **מינימום** – חתך שבין כל החתכים בגרף יש בו הכי מעט צלעות.

זאת, בניגוד לחתך **מינימלי** – חתך שאם נוריד ממנו אפילו עוד צלע אחת הוא לא יהיה חתך.

חוקיות הפתרון מהשיעור שעבר:

נניח כי בסיום הלולאה נותרו הקודקודים v_S ו-$v_{S'}$ כאשר S היא קבוצת כל הקודקודים שכיווצנו עד לכדי הקודקוד הראשון ו-S' היא קבוצת כל הקודקודים שכיווצנו עד לכדי הקודקוד השני.

כל צלע שהייתה במקור מ-S ל-S' נשמרה עד ל-v_S ו-$v_{S'}$.

כמו כן, כל צלע בין v_S ל-$v_{S'}$ הייתה במקור מ-S ל-S' (המשפט הזה פחות חשוב, הקודם הוא העיקר).

אופטימליות (כלומר, למה הוא מחזיר חתך מינימלי בהסתברות מספיק טובה):

נסמן ב-C חתך מינימום כלשהו ב-G, מולטי-גרף הקלט.

נניח כי $K = |C|$. נחשב את ההסתברות שהאלגוריתם מחזיר את הגודל K:

$$P\begin{pmatrix}\text{האלגוריתם מחזיר}\\ |C| = K \text{ את}\end{pmatrix} \geq P\begin{pmatrix}\text{לא הוסרה}\\ \text{אף צלע מ } C -\end{pmatrix}$$

נגדיר את סדרת המאורעות:

A_i – באיטרציה ה-i לא נבחרה צלע מ-C.

נרצה להראות כי מתקיים:

$$P\begin{pmatrix}\text{לא הוסרה}\\ \text{אף צלע מ } C -\end{pmatrix} \geq P\left(\bigcap_{i=1}^{n-2} A_i\right)$$

(יש לנו n קודקודים ואנחנו מאחדים את כולם חוץ מהשניים האחרונים, לכן יש n-2 חיתוכים)

לשם כך נטען את הטענה הבאה:

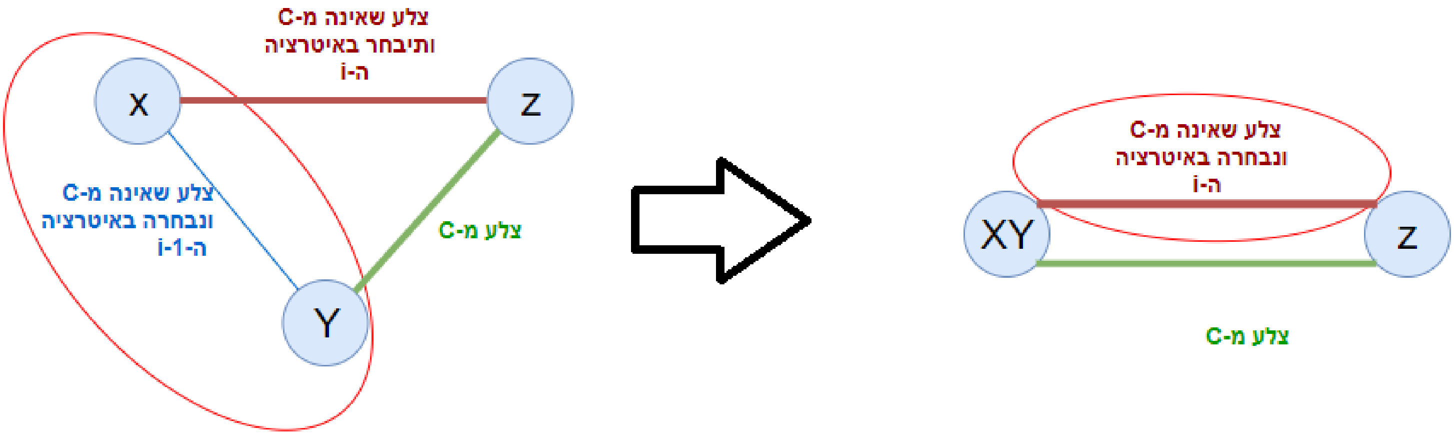
טענה:

אם באיטרציה ה-i לא **נבחרה** צלע מ-C אז גם בוודאות לא **הוסרה** צלע מ-C.

<u>הוכחה:</u>

נניח בשלילה כי בחרנו צלע a שאינה חלק מ-C וכתוצאה מאיחוד שני הקודקודים שבקצותיה הוסרה גם צלע b שהיא כן חלק מ-C (ונניח שעד לנקודה הזו לא הוסרו צלעות מ-C). אזי b לא הכרחית עבור C ואפשר להסירה ממנו, כי ההורדה שלה מהמולטי-גרף לא יוצרת נתק בין הקודקודים בקצותיה (יש מסלול ביניהם הבנוי מ-a וצלעות אחרות שלא נמצאות ב-C) – אולם זו סתירה לך ש-C חתך מינימום.

דוגמה:



Y,X ו-Z מהווים חלק מהמולטי-גרף G.

בדוגמה המתוארת Z ו-XY יאוחדו <u>ושתי</u> הצלעות תעלמנה, כלומר, יש איטרציה בה לא **נבחרה** צלע מ-C אך כן **הוסרה** צלע מ-C. זה סותר את הטענה שלנו! נראה שמצב זה לא אפשרי:

נבחין כי אם נסיר את C מהגרף, הסרת YZ לא מנתקת את Y ו-Z, שכן עדיין יש מסלול ביניהם המורכב כולו מצלעות שאינן ב-C ולכן לא הוסרו. כלומר, YZ לא חיונית לחתך ואפשר להוריד אותה ממנו – זאת, <u>בסתירה</u> לכך ש-C חתך מינימום.

נתחת את ההסתברות של כל המאורעות A_i:

$$\mathbf{P}(\overline{\mathbf{A_1}}) = \frac{K}{|E|} = \frac{K}{\frac{1}{2}\sum_{v\in V}\deg(v)} = 2\cdot\frac{K}{\sum_{v\in V}\deg(v)} \le 2\cdot\frac{K}{\sum_{v\in V}K} = \frac{2K}{nK} = \frac{2}{n}$$

הסברים למעברים:

1. יש לנו K צלעות של C מתוך $|E|$ צלעות של G.
2. נזכר כי מספר הצלעות בגרף היא מחצית סכום דרגות קודקודיו, כי אנחנו סופרים כל צלע פעמיים – פעם כשהיא יוצאת מקודקוד אחד ופעם כשהיא יוצאת מהשני.
3. הוצאת $\frac{1}{2}$ מהמכנה.

4. דרגת כל קודקוד היא לפחות K. נניח בשלילה שיש קודקוד עם דרגה קטנה מ-K, אז אפשר לנתק אותו מכל שכניו באמצעות הסרת פחות מ-K צלעות ולהגדיל את מספר רכיבי הקשירות באחד - כלומר, הצלעות בינו לבין שכניו הן חתך עם פחות מ-K צלעות, בסתירה לכך שבחתך עם הכי מעט צלעות יש K צלעות.
5. יש n קודקודים.

לכן:

$$\boxed{\mathbf{P}(\mathbf{A_1}) \geq 1 - \frac{2}{n}}$$

כמו כן, מתקיים:

$$\mathbf{P}(\overline{\mathbf{A_2}}|\boldsymbol{A_1}) = \frac{\mathrm{K}}{|\mathrm{E}| - 1} = \frac{\mathrm{K}}{\frac{1}{2}\sum_{\mathrm{v}\in \mathrm{V}_2} \deg(\mathrm{v})} \leq \frac{2\mathrm{K}}{(\mathrm{n}-1)\mathrm{K}} = \frac{2}{\mathrm{n}-1}$$

לכן:

$$\boxed{\mathbf{P}(\mathbf{A_2}|\boldsymbol{A_1}) \geq 1 - \frac{2}{\mathrm{n}-1}}$$

ובאופן כללי:

$$\mathbf{P}(\overline{\mathbf{A_{i+1}}}|\mathbf{A_1}, \dots, \mathbf{A}_{\boldsymbol{i}}) \leq \frac{2}{\mathrm{n}-\mathrm{i}}$$

נסיק אינדוקטיבית:

$$\mathbf{P}\left(\bigcap_{\mathbf{i}=\mathbf{1}}^{\mathbf{n}-\mathbf{2}} \mathbf{A_i}\right) = \mathrm{P}(\mathrm{A}_1)\cdot \mathrm{P}(\mathrm{A}_2|\mathrm{A}_1)\cdot \mathrm{P}(\mathrm{A}_3|\mathrm{A}_1,\mathrm{A}_2)\dots \geq \left(1-\frac{2}{\mathrm{n}}\right)\cdot\left(1-\frac{2}{\mathrm{n}-1}\right)\cdot\dots\left(1-\frac{2}{3}\right)$$

$$= \frac{\mathrm{n}-2}{\mathrm{n}}\cdot\frac{\mathrm{n}-3}{\mathrm{n}-1}\cdot\dots\frac{1}{3} = \frac{(\mathrm{n}-2)!}{\mathrm{n}\cdot(\mathrm{n}-1)\cdot\dots 3} = \frac{(\mathrm{n}-2)!}{\left(\frac{\mathrm{n}\cdot(\mathrm{n}-1)\cdot\dots 3\cdot 2\cdot 1}{2}\right)}$$

$$= \frac{(\mathrm{n}-2)!}{\frac{\mathrm{n}!}{2}} = 2\cdot\frac{(\mathrm{n}-2)!}{\mathrm{n}\cdot(\mathrm{n}-1)\cdot(\mathrm{n}-2)!} = \frac{2}{\mathrm{n}\cdot(\mathrm{n}-1)} > \frac{\mathbf{2}}{\mathbf{n^2}}$$

נחבר את כל החלקים ונראה שקיבלנו חסם תחתון על ההסתברות שהאלגוריתם יחזיר K, גודל חתך מינימלי בגרף:

$$\mathrm{P}\left(\begin{matrix}\text{האלגוריתם מחזיר}\\ |\mathrm{C}| = \mathrm{K}\ \text{את}\end{matrix}\right) \geq \mathrm{P}\left(\begin{matrix}\text{לא הוסרה}\\ \mathrm{C} - \text{אף צלע מ}\end{matrix}\right) \geq \mathrm{P}\left(\bigcap_{\mathrm{i}=1}^{\mathrm{n}-2}\mathrm{A_i}\right) > \frac{\mathbf{2}}{\boldsymbol{n^2}}$$

אז אנחנו יודעים שהאלגוריתם טועה בהסתברות טועה בהסתברות $\leq 1 - \frac{2}{n^2}$.

נריץ אותו $\frac{n^2}{2}$ פעמים וניקח את התוצאה המינימלית מבין כל התוצאות שהוחזרו בכל ההרצות.

ההסתברות שהוחזר גודל חתך שאינו מינימום = ההסתברות שהייתה טעות בכל ההרצות (כאשר אנחנו יודעים שכל ההרצות הן בלתי תלויות זו בזו) $\geq \left(1-\frac{2}{n^2}\right)^{\frac{n^2}{2}} > \frac{1}{e}$ (ההסתברות שואפת משמאל ל-$\frac{1}{e}$).

לכן, ההסתברות שהאלגוריתם צודק $\leq 1-\frac{1}{e}$.

היינו יכולים להריץ את האלגוריתם יותר פעמים ולקבל חסם טוב יותר, בפרט, אם היינו מריצים את האלגוריתם $\omega(n^2)$ פעמים היינו מקבלים שהאלגוריתם צודק בהסתברות ששואפת ל-1 עבור קלטים מספיק גדולים.

פתרון שאלה 1 מתרגיל 2

1. **המשחק פקמן**:

מפתחי המשחק פקמן פיתחו גרסא חדשה למשחק. בגרסא זו, הדמות זזה על גבי הסריג הדו־מימדי במאוזן או במאונך, ו"אוכלת" נקודות המפוזרות בו. עם זאת, כעת המטרה היא "לאכול" את הנקודות לפי סידור נתון מראש שלהן. כמו כן, הדמות יכולה "לאכול" נקודה על ידי עמידה על אחד הצירים שעליו הנקודה נמצאת, כלומר על הדמות להימצא באותה קואורידנטת x או באותה קואורדינטת y של הנקודה.

מפתחי המשחק ארגנו תחרות תכנות, בה המטרה היא לפתח אלגוריתם שבהינתן מיקום הנקודות והסדר שלהן, מנצח במשחק תוך מספר מינימלי של צעדים. פורמלית, הקלט הוא סדרת n נקודות במישור $(x_1,y_1),\dots,(x_n,y_n)$ והפלט הוא המספר המינימלי של צעדים שעל הדמות שבראשית הצירים לעבור כדי "לאכול" את n הנקודות לפי הסדר.

לפניכם מוצעת טענה: לכל קלט $(x_1,y_1),\dots,(x_n,y_n)$ קיים מסלול אופטימלי $p_1,\dots,p_n$ כך שכל רישא באורך i שלו היא מסלול אופטימלי לרישא באורך i של הקלט, לכל i. הנקודה p_k בתיאור זה של המסלול היא זו שממנה הדמות "אוכלת" את הנקודה ה־k בקלט, לכל k.

אם לדעתכם הטענה נכונה, הוכיחו את נכונותה והציעו אלגוריתם יעיל לבעיה המבוסס עליה, כלומר שזמן ריצתו פולינומי. אם לדעתכם היא לא נכונה, תארו דוגמא נגדית, והציעו אלגוריתם כלשהו, לא בהכרח יעיל, עבור הבעיה. הסבירו בכל מקרה נכונות וזמן ריצה.

בונוס: פתרונות שזמן ריצתם הוא **טוב יותר** מ־$O(n^2)$ תחת **ההנחה** כי כל נקודות הקלט ממוקמות בתת־ריבוע של הסריג, המכיל $O(n)$ נקודות מהסריג, יזכו ב־10 נקודות בונוס. פתרונות בעלי זמן ריצה כזה שלא מסתמכים על הנחה זו, יזכו ב־40 נקודות. הסבירו בקצרה נכונות וזמן ריצה.

פתרון 1 – עובד ב-$O(n^2)$:

הבחנות:

1. קיים מסלול אופטימלי בו כל מעבר מאכילת הנקודה i לאכילת הנקודה ה-i+1 הוא לאורך אחד הצירים. אם הוא כולל תנועה לאורך שני הצירים, ניתן לדחות אחד מהם לאיטרציה הבאה.

2. קיים מסלול אופטימלי בו לכל i אכילת הנקודה ה-i, שנסמנה (x_i,y_i), מהנקודה (x,y) מקיימת $x=x_i$ וכן y הוא ערך קואורדינטת ה-y של אחת הנקודות הקודמות במסלול או להפך (כלומר, $y=y_i$ וגם x הוא ערך קואורדינטת ה-x של אחת הנקודות הקודמות במסלול).

נגדיר טבלת תכנון דינאמי שתוגדר באמצעות הנוסחה:

$$f(i,k) = \boxed{\begin{array}{c} f_x(i,k) \\ \hline f_y(i,k) \end{array}}$$

- $f_x(i,k)$ – אורך המסלול האופטימלי עד לאכילת הנקודה ה-i דרך $(\boldsymbol{x_i}, \boldsymbol{y_k})$.

  (כלומר אוכלים את הנקודה ה-i דרך קואורדינטת ה-x).
- $f_y(i,k)$ - אורך המסלול האופטימלי עד לאכילת הנקודה ה-i דרך $(\boldsymbol{x_k}, \boldsymbol{y_i})$.

  (כלומר אוכלים את הנקודה ה-i דרך קואורדינטת ה-y).

נגדיר את אופן מילוי הטבלה:

$$f_x(i,k) = \begin{cases} |x_1| & i = 1 \\ f_x(i-1,k) + |x_i - x_{i-1}| & i > 1, 0 \le k < i-1 \\ \min\limits_{j<k-1} \{f_y(i-1,j) + |x_j - x_i|\} & i > 1, k = i-1 \end{cases}$$

הגדרת אופן מילוי הטבלה עבור $f_y(i,k)$ הוא סימטרי.

נסביר כל אחד מהמקרים:

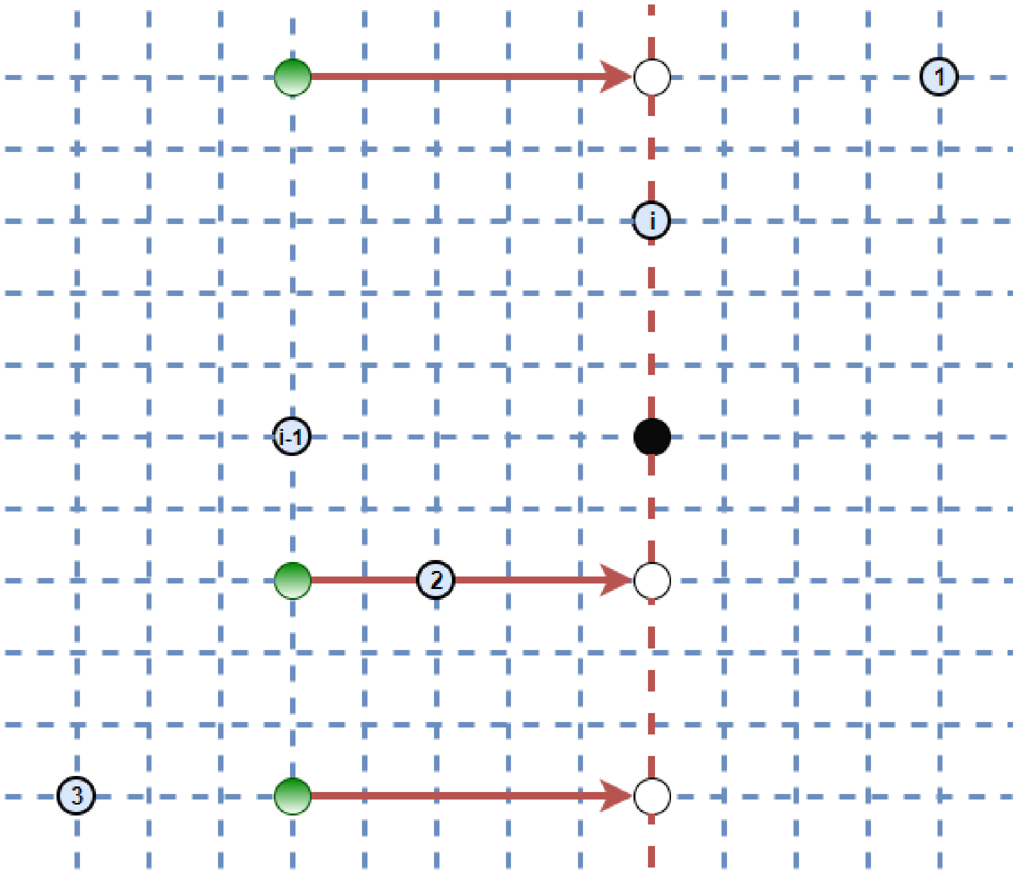
1. אנחנו רוצים לאכול את הנקודה הראשונה דרך קואורדינטת ה-x שלה, לכן צריך להתקדם לאורך ציר ה-x מראשית הצירים $|x_1|$ צעדים.

2. אנחנו רוצים לאכול את הנקודה ה-i, שאינה הראשונה, דרך קואורדינטת ה-x שלה, כאשר ערך ה-y של הנקודה ממנה נאכל שווה לערך של נקודה קודמת שאכלנו כבר (נקרא לה (x_k, y_k)) אבל **אינה הנקודה ה-i-1**. לפי הבחנה 1 אנחנו רוצים לזוז רק בציר אחד, הלוא הוא ציר ה-x במקרה שלנו[9], ולשם כך אנחנו רוצים להיות כבר בנקודה עם ערך y ששווה ל-y_k טרם אכילת הנקודה ה-i. במילים אחרות, היינו רוצים לאכול את הנקודה ה-i-1 (באמצעות מסלול אופטימלי) מנקודה בעלת ערך y שהוא y_k ובעלת ערך x שהוא x_{i-1} ואז להתקדם את

[9] כי אם נזוז רק בציר ה-y ונאכל את הנקודה ה-i דרך ציר ה-x סימן שכבר היינו בקואורדינטת ה-x הנכונה וזזנו לשווא לאורך ציר ה-y. במקרה זה עלינו להישאר במקום – כלומר לזוז 0 בציר ה-x ו-0 בציר ה -y ואפשר להתייחס לזה כתזוזה של 0 צעדים דווקא בציר ה-x.

הצעדים ההכרחיים לאורך ציר ה-x כך שנאכל את הנקודה ה-i מהנקודה (x_i, y_k). זה בדיוק הביטוי $f_x(i-1,k) + |x_i - x_{i-1}|$.

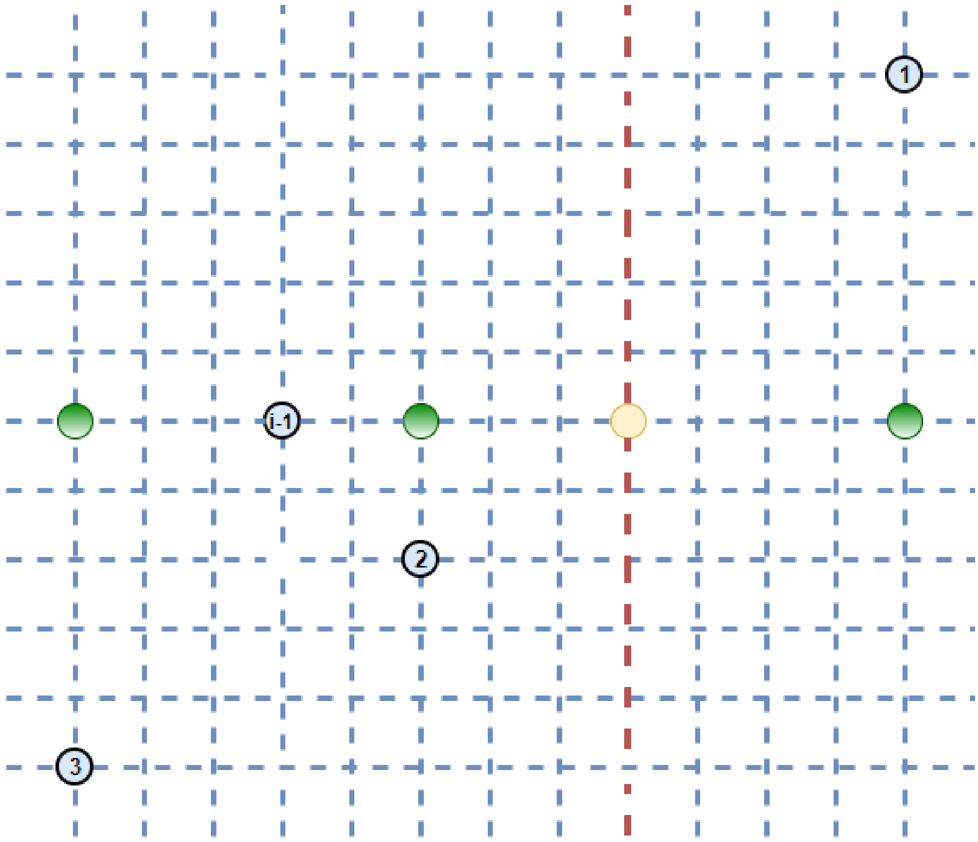
באיור להלן ניתן לראות את הנקודות הראשונה, השנייה והשלישית שאכלנו מסומנות ב-1,2,3 בהתאמה. כמו כן רואים את הנקודות ה-i וה-i-1 שצריך לאכול. כל הנקודות מהן אפשר לאכול את הנקודה ה-i מסומנות בלבן וזו שאי אפשר לאכול ממנה מסומנת בשחור (כלומר, אפשר לאכול ממנה, אבל התרחיש הזה נופל תחת התנאי השלישי של נוסחת מילוי הטבלה). הנקודות המלאות חלקית בצבע ירוק מסמנות את הנקודות מהן ניתן לאכול את הנקודה ה-i-1 דרך ציר ה-x לפי התיאור שלנו, נבחר את זו שהמסלול אליה אופטימלי (באמצעות הערך $f_x(i-1,k)$) ונוסיף לה את הערך $|x_i - x_{i-1}|$.



3. אנחנו רוצים לאכול את הנקודה ה-i, שאינה הראשונה, דרך קואורדינטת ה-x שלה, כאשר ערך ה-y של הנקודה ממנה נאכל שווה לערך של הנקודה האחרונה שאכלנו. מאותו הגיון כמו במקרה 2, אנחנו רוצים להתקדם רק בציר ה-x ולכן היינו רוצים לאכול את הנקודה ה-i-1 מנקודה כלשהי שערך ה-y שלה הוא y_{i-1}, כלומר, נרצה לאכול את הנקודה ה-i-1 דרך קואורדינטת ה-y שלה ואז להתקדם את מספר הצעדים ההכרחיים לאורך ציר ה-x. אבל מה

ערך ה-x של הנקודה ממנה נאכל את הנקודה ה-i-1? נסמן את הערך הזה ב-x_j[10] ועכשיו נחפש את הנקודה (x_j, y_{i-1}) שהמסלול אליה ועוד המסלול ממנה אל $(\mathrm{x_i}, y_{i-1})$ הוא הכי קצר. ובנוסחה: $\min_{\mathrm{j<k-1}}\{f_y(i-1,j)+|x_j-x_i|\}$.

באיור להלן ניתן לראות את הנקודות הראשונה, השנייה והשלישית שאכלנו מסומנות ב-1,2,3 בהתאמה. כמו כן, ניתן לראות את הנקודה (x_i, y_{i-1}) הצהובה ממנה אנחנו רוצים לאכול את הנקודה ה-i. הנקודות הירוקות הן אלו אשר מהן אנחנו יכולים לאכול את הנקודה ה-i-1 כך שנוכל לזוז אחר רק לאורך ציר ה-x ולהגיע אל הנקודה הצהובה. נבחר, מבין כל הנקודות הירוקות, את אלו שאורך המסלול האופטימלי אליהן ועוד המרחק מהן אל הנקודה הצהובה הוא מינימלי, כלומר $\min_{\mathrm{j<k-1}}\{f_y(i-1,j)+|x_j-x_i|\}$.



הערך של האלגוריתם הוא המינימום על פני כל התאים שיש בשורה האחרונה בטבלה.
התא ה-j בה מחזיק את המסלול הכי קצר שאוכל את כל הנקודות ומסתיים ב-(x_n, y_j) וערך נוסף בחלקו השני - כזה שמסתיים ב-(x_j, y_n).
לפי אבחנה 2 אלו הנקודות היחידות שצריך לבדוק, ולכן המינימלית על פניהם היא המבוקשת.

---

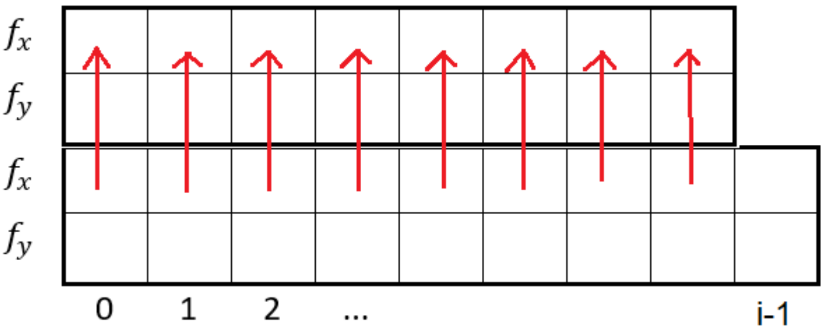
[10] כלומר, מדובר בנקודה עם ערך x של הנקודה ה-j שאכלנו.

זמן ריצה - $O(n^2)$:

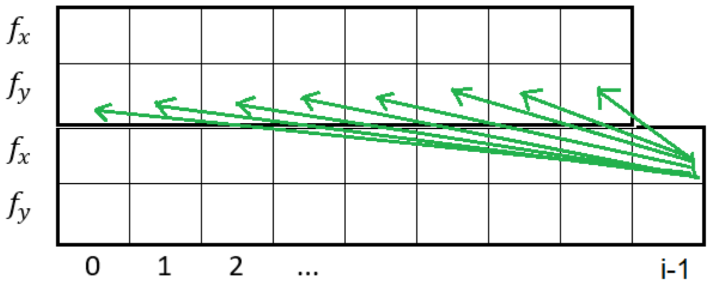
ננתח את מילוי השורה ה-i בטבלה (שוב נסתכל על מילוי f_x אך מילוי f_y דומה מאוד):

עבור כל התאים משמאל לתא ה-i-1 - פשוט בודקים את ערך ה-f_x באותו התא בשורה ה-i-1 ומוסיפים ערך מוחלט שקל לחשב. כלומר מילוי כל תא כזה עולה $O(1)$.

נדגיש כי אחת הנקודות אליהן מתייחסים היא הנקודה $(0,0)$, על אף שאינה חלק מהקלט ולכן העמודה הראשונה מוקדשת עבורה.



עבור התא ה-i-1 – עוברים על כל ערכי ה-f_y בשורה ה-i-1, מוסיפים לכל ערך כזה מספר שקל לחשב $\left(|x_j - x_i|\right)$ ומוצאים את המינימום על כל הערכים האלו. כלומר, מילוי התא ה-i-1 עולה לנו $O(n)$.



בסך הכל, בכל שורה יש לנו $O(n)$ תאים שאנחנו ממלאים בעלות $O(1)$ ועוד תא אחד שאנחנו ממלאים בעלות של $O(n)$. כלומר, מילוי שורה אחת עולה לנו $O(n \cdot 1 + n) = O(n)$.

יש לנו $O(n)$ שורות ולכן העלות הכוללת של מילוי הטבלה היא: $\boldsymbol{O(n^2)}$.

שיעור 9 – 14.12.2020

בחלק הראשון של השיעור חזרנו על הפתרון הראשון של שאלת הבונוס מתרגיל הבית השני, תוכן השיעור נמצא בסיכום שיעור 8.

בעיית מנוע החיפוש

בהינתן מחרוזת ארוכה T עם $|T| = n$ מעוניינים לענות על שאילתות מהצורה "האם המחרוזת הקצרה P נמצאת כתת-מחרוזת רצופה ב-T?" עבור מחרוזת P עם $|P| = k$.

פתרון נאיבי:

לכל אות ב-T בודקים האם P מופיעה החל ממנה ומחזירים "כן" אם כן.

דוגמה:

$$\boxed{\begin{array}{c}\mathbf{P = ABD} \\ \mathbf{T = ABCABD}\end{array}}$$

בודקים האם P מתחילה בכל אות ב-T באמצעות השוואה אות-אות.

כשמגיעים ל-A השנייה ב-T רואים כי $A == A, B == B, D == D$ ומחזירים "כן".

זמן ריצה (של השאילתה): $O(n \cdot k)$.

עבור כל אחת מהאותיות ב-T בודקים k אותיות קדימה ומשווים אותן אות-אות עם P.

פתרון 1 (עיבוד מוקדם ב-$O(n^2 \log(n))$, מענה על שאילתה ב-$O(k \log(n))$)

- **עיבוד מוקדם:**

  הגדר את B, מערך מחרוזות הסיפות של T ומיין אותו לקסיקוגרפית.

  כלומר, ניקח את כל הסיומות של T, נשים אותן במערך ונמיין לקסיקוגרפית. למשל, עבור הדוגמה:

$$\boxed{\mathbf{B = [ABCABD, ABD, BCABD, BD, CABD, D]}}$$

- **מענה על שאילתה:**

  בצע חיפוש בינארי של המחרוזת P במערך והחזר "כן" אם נמצאה כרישא של מילה בו.

  בדוגמה שלנו נבדוק תחילה את BCABD, נראה כי B גדולה מ-A ונדע כי עלינו להמשיך לחפש משמאל ל-BCABD. בשלב הבא נגיד שנבדוק את ABCABD – נראה כי A מתאימה, B מתאימה אך C לא. משום ש-C קטנה מ-D נדע שעלינו לבדוק ימינה מ-ABCABD במערך, לכן נבדוק את ABD, נראה ש-ABD אכן רישא שלה ונחזיר "כן".

זמן ריצה:

- **עיבוד מוקדם - $O(n^2 \log(n))$:**

למשל כעלות מיון מיזוג[11], כשכל השוואה בו (בחלק של המיזוג) בעלות $O(n)$ במקום $O(1)$ (כי משווים מחרוזות ולא, למשל, מספרים טבעיים) – לכן העלות היא $O(n^2 \cdot \log(n))$.

- **מענה על שאילתה - $O(k \cdot \log(n))$:**

מספר האיברים הנסרקים הוא $O(\log(n))$ וכל השוואה בעלות $O(k)$ (כי כשבודקים האם P היא רישא של מחרוזת, צריך לבדוק רק את $|P| = k$ האותיות הראשונות של המחרוזת).

פתרון 2 -(עיבוד מוקדם ב-$O(n)$, מענה על שאילתה ב-$O(k + \log(n))$)

- **עיבוד מוקדם:**

מערך הסיפות של מחרוזת T שיסומן ב-SA(T) (עבור Suffix Array) הוא המערך בו באינדקס ה-i נמצא <u>המיקום</u> (של האות הראשונה) של הסיפא ה-i של T.

עבור הדוגמה שלנו נזכר כי:

$$\mathbf{T = ABCABD}$$

$$\mathbf{B = [ABCABD, ABD, BCABD, BD, CABD, D]}$$

ונקבל:

$$\boxed{\mathbf{SA(T) = [1, 4, 2, 5, 3, 6]}}$$

המערך בגודל $O(n)$ ואפשר למיין אותו ב-$O(n)$ באמצעות <u>אלגוריתם</u> שמזכיר את radix-sort.

קיבלנו, אם כן, הצגה חסכונית של B - יש לנו את אותו המידע בפחות מקום:

נשים לב כי נוכל למצוא את הסיפא ה-i ב-B באמצעות הביטוי $\mathrm{T[SA[i], \ldots, n]}$.

למשל, אם נרצה בדוגמה שלנו למצוא את הסיפא הרביעית, נלך אל $\mathrm{SA[4]}$, נקבל 5, ניקח את $\mathrm{T[5,6]}$ ונקבל את BD, שהיא אכן הסיפא הרביעית ב-B.

<u>הגדרה – LCP:</u>

---

[11] נזכר כי במיון מיזוג לוקחים בכל איטרציה את המערך המקורי, מחלקים אותו לשניים, ממינים כל חלק ואז ממזגים את שני החלקים הממוינים. החלק היקר של מיון מיזוג הוא המיזוג – מבצעים אותו בעזרת ריצה על שני התת-מערכים עם שני מצביעים והשוואה בין איברים. הנוסחה של מיון מיזוג היא $\mathrm{T(n)} = 2\mathrm{T}\left(\frac{\mathrm{n}}{2}\right) + \theta(\mathrm{n}) = n \cdot \log(n)$ (בהנחה כי $\mathrm{T(1)} = 1$).

נגדיר את ה-LCP (Longest Common Prefix) של זוג מחרוזות s_1, s_2 **כאורך** הרישא המשותפת המקסימלית שלהן. למשל: $\text{LCP}(\text{ABC}, \text{ABD}) = 2$.

בעיבוד המוקדם, נרצה לייצר ביעילות מבנה נתונים שמאפשר למצוא LCP של סיפות של T ביעילות.

<u>הערה</u>: אם מחרוזת היא רישא של אחרת אז הארוכה תופיע לפני הקצרה.
למשל, אם היו לנו שתי מחרוזות ABD ו-AB, אז ABD תהיה קודמת ל-AB.

<u>הקדמה לאופן המענה על שאילתה (נתאר אותו בהרחבה בשבוע הבא):</u>
נשים לב שאפשר להשתמש ב-LCP כדי לבצע חיפוש בינארי על SA(T) בצורה דומה לזו המתוארת בפתרון 1: כשמשווים את P לאחת הסיפות ב-SA(T) (כלומר, אחת הסיפות ב-B ששחזרנו באמצעות SA(T) ו-T), אם נריץ LCP על שתיהן ונקבל k נדע כי מצאנו סיפא מתאימה. אחרת, נדע מהר (מבלי להשוות את כל האותיות הראשונות) איזו אות בסיפא היא הראשונה ששונה ממקבילתה ב-P, נשווה ב-$O(1)$ בינה לבין מקבילתה ונדע לאן להתקדם ב-SA(T) (אם האות בסיפא גדולה מזו ב-P נתקדם שמאלה, אחרת ימינה).

אם הייתה לנו דרך לבצע LCP ב-$O(1)$ היינו מקבלים השוואה בין שתי מחרוזות שעולה לנו $O(1)$ במקום $O(k)$ וזמן המענה על שאילתה היה מתכווץ מ-$O(k \cdot \log n)$ ל-$O(\log n)$.
הבעיה היא שלא סביר שיש "קופסה שחורה" כלשהי שיכולה למצוא LCP של כל מחרוזת עם כל מחרוזת אחרת ב-$O(1)$, היינו רוצים להכין אותה בשלב העיבוד המוקדם כדי לענות על שאילתות ביעילות אבל לא סביר שנדע מה ה-LCP של שאילתה כלשהי עם כל הסיפות ב-SA(T) (או ליתר דיוק ב-B). בשבוע הבא נראה שבמקום זה אפשר לשמור את ה-LCP של כל זוג סיפות ב-SA(T), להיעזר במידע הזה בחיפוש הבינארי ובכך לחסוך את רוב ההשוואות בין P למחרוזות ב-SA(T).

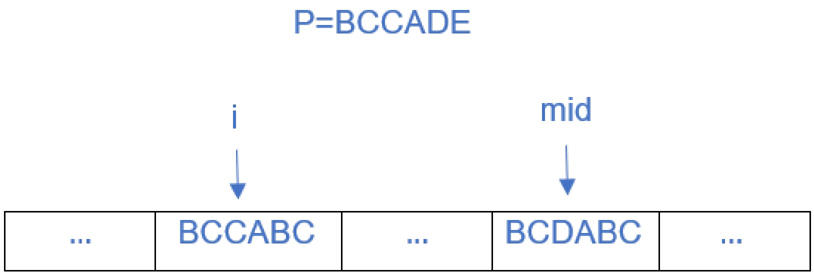
שיעור 10 – 21.12.2020

בעיית מנוע החיפוש, פתרון 2 – המשך:

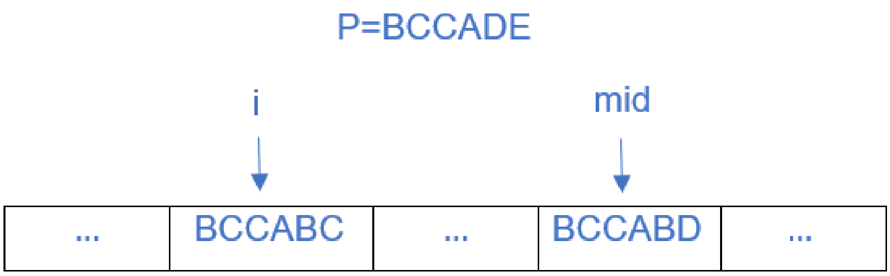
- **מענה על שאילתה:**

1. אתחל מונה $c = 0$ ו-$i = n + 1$ כך ש-i הוא האינדקס במערך הסיפות של הסיפא הדומה ביותר ל-P שנמצאה עד כה ו-c היא מידת הדמיון ביניהן (כלומר, ה-LCP של שתיהן)
(בהתחלה המחרוזת הדומה ביותר ל-P מוגדרת להיות סיפא ריקה, מעין null, לכן c=0).

2. בצע כל עוד הקטע הנסרק במערך הסיפות באורך אי-שלילי[12]:
   א. אם $\mathrm{LCP}(\mathrm{T}[\mathrm{SA}[\mathrm{mid}], \dots, n], \mathrm{T}[\mathrm{SA}[\mathrm{i}], \dots, n]) < \mathrm{c}$ התקרב ל-i (הזז את mid לפי חיפוש בינארי).
   בדוגמה, $\mathrm{LCP}(\mathrm{BCCADE}, \mathrm{BCCABC}) = 4 = \mathrm{c}$ ואילו $\mathrm{LCP}(\mathrm{BCCABC}, \mathrm{BCDABC}) = 2$, לכן ניתן להסיק כי BCDABC וכל הסיפות מימינה דומות ל-P פחות מאשר הסיפא ה-i, לכן לא נתקדם ימינה ונתקדם שמאלה, כלומר, נתקרב אל i. (נשים לב כי $\mathrm{T}[\mathrm{SA}[\mathrm{mid}], \dots, \mathrm{n}] = \mathrm{BCDABC}$ וגם $\mathrm{T}[\mathrm{SA}[\mathrm{i}], \dots, \mathrm{n}] = \mathrm{BCCABC}$).



   ב. אם $\mathrm{LCP}(\mathrm{T}[\mathrm{SA}[\mathrm{mid}], \dots, \mathrm{n}], \mathrm{T}[\mathrm{SA}[\mathrm{i}], \dots, \mathrm{n}]) > \mathrm{c}$, התרחק מ-i.
   בדוגמה רואים כי c=4, אך $\mathrm{LCP}(\mathrm{BCCABC}, \mathrm{BCCABD}) = 5$. כלומר, כל הסיפות בין הסיפא ה-i לסיפא ה-mid דומות יותר לסיפא ה-i מאשר הסיפא ה-mid, לכן אין מה לחפש שם ונתקדם ימינה מ-mid, כלומר, נתרחק מ-i.



---

[12] בחיפוש בינארי יש לנו שני מצביעים: right ו-left ומפסיקים את החיפוש רק כאשר right עובר להיות משמאל ל-left.

ג. אם $\mathrm{LCP}(\mathrm{T}[\mathrm{SA}[\mathrm{mid}], \dots, \mathrm{n}], \mathrm{T}[\mathrm{SA}[\mathrm{i}], \dots, \mathrm{n}]) = \mathrm{c}$, השווה את P למחרוזת ה-mid החל מהתו ה-c+1 (הסבר[13]) ועדכן את i ו-c בהתאם[14]. אם $\mathrm{c} = |\mathrm{P}|$ החזר "כן" (כי P נמצאת ברישא של המחרוזת) - אחרת, עדכן את mid לפי האות השונה (אם היא גדולה יותר בסיפא ה-mid, נזיז את mid שמאלה ואחרת ימינה).

3. החזר "לא" (c אף פעם לא מגיעה ל-|P|, לכן P לא תת-מחרוזת של אף סיפא של T ולכן אינה תת-מחרוזת של T).

זמן ריצה של פתרון 2 - $O(k + \log(n))$:

- התקדמות בחיפוש הבינארי עולה $O(\log(n))$.
- סריקת P עולה $O(k)$.

נכונות האלגוריתם:

האלגוריתם בנוי משלושה שלבים, בכל אחד משווים את הסיפא הנוכחית (הסיפא ה-mid) אל ה-argmax (הסיפא שעד עכשיו הייתה הכי דומה ל-P, כלומר, הסיפא ה-i) ואז:

א. רואים שהן שונות מדי (P דומה יחסית ל-argmax הנוכחי, אבל הנוכחית כבר די שונה מה-argmax), זה אומר שהתרחקנו יותר מדי מה-argmax הנוכחית ולכן צריך להתקרב אליה.

ב. רואים שהן דומות מדי, זה אומר שלא התרחקנו מספיק מה-argmax הנוכחית ולכן צריך להתרחק ממנה יותר.

ג. רואים שהן דומות באותה מידה כמו ש-P וה-argmax דומות, לכן אפשר לקרוא לסיפא הנוכחית argmax (כלומר להגדיר את mid להיות ה-i החדש). בנוסף, נרצה להשוות ישירות בין המחרוזת הנוכחית (שתוגדר כ-argmax) ל-P באופן ישיר, אך אנחנו יודעים כי צריך להשוות רק החל מהתו ה-c+1.

כעת, נחזור לעיבוד המוקדם כדי לסיים כמה נושאים שהשארנו פתוחים:

---

[13] אנחנו יודעים ש-c האותיות הראשונות של P ושל הסיפא ה-i זהות וגם ש-c האותיות הראשונות של הסיפא ה-i ושל הסיפא ה-mid זהות. לכן, c האותיות הראשונות של P ושל הסיפא ה-mid זהות ואין טעם להשוות ביניהן.

[14] $i \leftarrow mid$ וגם $\mathrm{c} \leftarrow \mathrm{c} + \mathrm{LCP}(\mathrm{T}[\mathrm{mid} + \mathrm{c}, \dots, \mathrm{n}], \mathrm{P}[\mathrm{c} + 1, \dots, \mathrm{k}])$, כלומר, נוסיף ל-c את כמות האותיות הזהות בין P למחרוזת ה-mid החל מהאות ה-c+1 שלהן וגם המחרוזת ה-mid תסומן כמחרוזת ה-i.

- **עיבוד מוקדם:**

דרישות מהעיבוד המוקדם (שתשרתנה את המענה על שאילתה):

1. החזרה של המערך SA ב-$O(n)$ – אולי נספיק לדבר על זה בשיעור הבא.
2. מבנה נתונים שעונה על שאילתות LCP על סיפות של T ב-$O(1)$ ושאפשר יהיה לבנות אותו ב-$O(n)$.

נגדיר סימון חדש כדי להקל על הכתיבה: $\mathbf{LCP}(\mathbf{i},\mathbf{j}) = \mathrm{LCP}(\mathrm{T}[\mathrm{i},\dots,\mathrm{n}],\mathrm{T}[\mathrm{j},\dots,\mathrm{n}])$

<u>טענת פונקציית ה-LCP:</u>

$$\mathbf{LCP}(\mathbf{SA}[\mathbf{i}],\mathbf{SA}[\mathbf{j}]) = \min_{\mathbf{i}\le\mathbf{k}<\mathbf{j}} \mathbf{LCP}(\mathbf{SA}[\mathbf{k}],\mathbf{SA}[\mathbf{k}+\mathbf{1}])$$

<u>הערות</u>

- כלומר, ה-LCP של שתי סיפות ב-SA שווה ל-LCP המינימלי בין כל שתי סיפות עוקבות שביניהן.
- נשים לב שבתא ה-i של SA יש אינדקס ולכן הסימון $\mathrm{LCP}(\mathrm{SA}[\mathrm{i}],\mathrm{SA}[\mathrm{j}])$ לגיטימי.
- אין קשר בין k שרצה בין i ל-j לבין האורך של P שגם סימנו כ-k.

<u>הוכחה:</u>

נסמן את המינימום ב-m. ראשית, נראה כי $\mathrm{LCP}(\mathrm{SA}[\mathrm{i}],\mathrm{SA}[\mathrm{j}]) \ge m$:

$$\mathrm{T}[\mathrm{SA}[\mathrm{i}],\dots,\mathrm{SA}[\mathrm{i}]+\mathrm{m}-1] = \mathrm{T}[\mathrm{SA}[\mathrm{i}+1],\dots,\mathrm{SA}[\mathrm{i}+1]+\mathrm{m}-1] = \cdots =$$
$$\mathrm{T}[\mathrm{SA}[\mathrm{j}],\dots,\mathrm{SA}[\mathrm{j}]+\mathrm{m}-1]$$

הסבר: m האותיות הראשונות בין הסיפא ה-i לבין הסיפא שאחריה בוודאי זהות, כי m הוא ה-LCP המינימלי, כנ"ל לגבי הסיפא ה-i+1 הסיפא ה-i+2 וכן הלאה עד לסיפא ה-j-1 והסיפא ה-j.

הראינו כי <u>לפחות</u> m האותיות הראשונות בין הסיפא ה-i והסיפא ה-j זהות.

עכשיו נרצה להראות כי <u>רק</u> m האותיות הראשונות בין השתיים זהות. לפי המיון, מתקיים:

$$\mathrm{T}[\mathrm{SA}[\mathrm{i}]+\mathrm{m}] \le \mathrm{T}[\mathrm{SA}[\mathrm{i}+1]+\mathrm{m}] \le \cdots \le \mathrm{T}[\mathrm{SA}[\mathrm{j}]+\mathrm{m}]$$

המיון לקסיקוגרפי, לכן לכל זוג סיפות עוקבות יש שתי אפשרויות:

1. הערך של m+1 האותיות הראשונות בהן זהה.
2. הערך של m+1 האותיות הראשונות אצל הסיפא הראשונה מבין השתיים קטן מזה של m+1 הראשונות בשנייה.

בנוסף, מכיוון ש-m הוא המינימום, אי שוויון אחד מבין אלו שלעיל הוא בוודאי אי שוויון חזק, אחרת המינימום היה m+1 או יותר. נובע מכך כי $\mathrm{T}[\mathrm{SA}[\mathrm{i}]+\mathrm{m}] < \mathrm{T}[\mathrm{SA}[\mathrm{j}]+\mathrm{m}]$, כלומר, האות ה-m+1 בסיפא ה-i וה-j שונה.

ובמילים אחרות: $\mathrm{LCP}(\mathrm{SA}[\mathrm{i}],\mathrm{SA}[\mathrm{j}]) = \min_{\mathrm{i}\le\mathrm{k}<\mathrm{j}} \mathrm{LCP}(\mathrm{SA}[\mathrm{k}],\mathrm{SA}[\mathrm{k}+1])$, כנדרש. ■

נגדיר את מערך ה-LCP:

$$\boxed{\mathrm{LCPA}[k] = \mathrm{LCP}(\mathrm{SA}[k], \mathrm{SA}[k+1])}$$

כלומר, זה מערך שכל תא בו מכיל את ה-LCP בין שתי סיפות צמודות ב-B (מערך הסיפות).

הבחנה:

$$\mathbf{LCP(i,j)} = \mathrm{LCP}\left(\mathrm{SA}\left[\mathrm{SA}^{-1}[i]\right], \mathrm{SA}\left[\mathrm{SA}^{-1}[j]\right]\right) = \min_{\mathrm{SA}^{-1}[i] \le k < \mathrm{SA}^{-1}[j]} \mathrm{LCP}(\mathrm{SA}[k], \mathrm{SA}[k+1])$$

$$= \min_{\mathbf{SA^{-1}}[i] \le \mathbf{k} < \mathbf{SA^{-1}}[j]} \mathbf{LCPA[k]}$$

הערות:

1. נשים לב כי $\mathrm{LCP}(i,j)$ זה לא אותו הדבר כמו $\mathrm{LCP}(\mathrm{SA}[i], \mathrm{SA}[j])$!
   $\mathrm{LCP}(i,j)$: ה-LCP של שתי סיפות של T, אחת שמתחילה מהאינדקס ה-i של T והשנייה מהאינדקס ה-j.
   $\mathrm{LCP}(\mathrm{SA}[i], \mathrm{SA}[j])$: ה-LCP של שתי סיפות של T: הסיפה ה-i לפי המיון של B, והסיפה ה-j.
2. ההעתקה SA היא חח"ע ועל, לכן ניתן להגדיר את SA^{-1}. אז $\mathrm{SA}\left[\mathrm{SA}^{-1}[i]\right] = i$, לכן המעבר הראשון טריוויאלי.

אז אם נריץ שאילתת מינימום על קטע במערך LCPA נוכל לקבל LCP של שתי סיפות.

נגדיר עיבוד מקדים של RMQ5[15] על LCPA ונקבל מבנה נתונים שעונה על שאילתות LCP ב-$O(1)$, כפי שדרשנו בדרישה מספר 2 מהעיבוד המוקדם.

אבל בדרישה מספר 2 גם דרשנו שאפשר יהיה לבנות את LCPA ב-$O(n)$, נראה איך עושים זאת:

חישוב LCPA ביעילות:

טענת מערך ה-LCP:

$$\mathrm{LCPA}\left[\mathrm{SA}^{-1}[i+1]\right] \ge \mathrm{LCPA}\left[\mathrm{SA}^{-1}[i]\right] - 1$$

נזכר בהגדרה של LCPA, $\mathrm{LCPA}[k] = \mathrm{LCP}(\mathrm{SA}[k], \mathrm{SA}[k+1])$, ונקבל:

$$\mathrm{LCPA}\left[\mathrm{SA}^{-1}[i+1]\right] = \mathrm{LCP}\left(\mathrm{SA}\left[\mathrm{SA}^{-1}[i+1]\right], \mathrm{SA}\left[\mathrm{SA}^{-1}[i+1]+1\right]\right) = \mathrm{LCP}\left(i+1, \mathrm{SA}\left[\mathrm{SA}^{-1}[i+1]+1\right]\right)$$

אנחנו שואלים את עצמנו מה ה-LCP של הסיפא שמתחילה באינדקס ה-i+1 של T והסיפא שבאה אחריה **במיון** (**לא** הסיפא שמתחילה באינדקס ה-i+2 של T). לפי הטענה, אם נסתכל על הסיפא שמתחילה באינדקס ה-i, נראה מה ה-LCP שלה ושל הסיפא העוקבת לה במיון ונחסיר מזה 1, נקבל תוצאה שקטנה-שווה ל-LCP של הסיפא המתחילה מהאינדקס ה-i+1 וזו העוקבת לה במיון.

למשל, אם נתונה לנו הסיפא ABCABD והבאה אחריה במיון היא ABD, אנחנו יודעים שה-LCP של שתיהן הוא 2. אם נתקדם תו אחד בשתי הסיפות ונסתכל על הסיפא BCABD, יכול להיות שהסיפא העוקבת לה במיון תהיה BD ויכול להיות שתהיה עוד סיפא בין שתיהן – מה שבטוח הוא שה-LCP של BCABD ושל העוקבת לה יהיה לפחות 1.

---

[15] לא הרחבנו על RMQ5 במסגרת השיעור, הוא פועל בסיבוכיות של $(O(n), O(1), O(n))$.

<u>הוכחה:</u>

נסמן:

1) $m = LCPA\left[SA^{-1}[i]\right]$

סימנו כעת כמה הסיפא שמתחילה באינדקס i ב-T דומה לסיפא העוקבת לה במיון.

נניח כי $m \geq 1$, אחרת אנחנו אומרים כי m הוא חיובי וזה טריוויאלי.

2) $j = SA[SA^{-1}[i] + 1]$

ונקבל (לפי הגדרת LCPA):

$$m = LCPA\left[SA^{-1}[i]\right] = LCP\left(SA\left[SA^{-1}[i]\right], SA[SA^{-1}[i] + 1]\right) = LCP(i, j)$$

או במילים אחרות, שני הדברים הבאים מתקיימים:

$$(*) T[i, \dots, i + m - 1] = T[j, \dots, j + m - 1]$$

$$(**) T[i + m] \neq T[j + m]$$

כלומר, m התווים הראשונים בסיפות שמתחילות באינדקס ה-i וה-j זהות, אך התו ה-m+1 שלהן שונה.

מכאן נקבל:

$$(*) \Rightarrow T[i + 1, \dots, (i + 1) + (m - 1) - 1] = T[j + 1, \dots, (j + 1) + (m - 1) - 1]$$

$$(**) \Rightarrow T[i + 1 + (m - 1)] \neq T[j + 1 + (m - 1)]$$

מסקנה: ניקח את הסיפא המתחילה בתו ה-i של T ונביט בסיפא העוקבת לה לפי המיון – נסמנה כסיפא ה-j. אם נסתכל על הסיפא שמתחילה מהתו ה-i+1 ועל זו שמתחילה תו אחד אחרי תחילת הסיפא ה-j נקבל כי ה-LCP שלהן הוא m-1.

ולסיום:

$$LCPA\left[SA^{-1}[i + 1]\right] \geq \min_{SA^{-1}[i+1] \leq k < SA^{-1}[j+1]} LCPA[k] = LCP(i + 1, j + 1) = m - 1$$
$$= LCPA\left[SA^{-1}[i]\right] - 1$$

<u>הסבר של המעברים:</u>

1. לפי הגדרה, $LCPA\left[SA^{-1}[i + 1]\right]$ הוא ה-LCP בין הסיפא ה-$SA^{-1}[i + 1]$ לבין זו העוקבת לה לפי המיון.
   הביטוי $\min_{SA^{-1}[i+1] \leq k < SA^{-1}[j+1]} LCPA[k]$ בודק עבור כל הסיפות החל מהסיפא ה-$SA^{-1}[i + 1]$ ועד לסיפא ה-$SA^{-1}[j + 1]$ מה ה-LCP שלה עם העוקבת לה ואז לוקח את הערך המינימלי. בפרט, ה-LCP בין הסיפא ה-$SA^{-1}[i + 1]$ והעוקבת לה נבדק (אלו שתי הסיפות הראשונות בקטע) ועל כן הוא גדול שווה מהמינימום.
2. את המעבר הזה ניתן להסביר באמצעות ההבחנה מהעמוד הקודם.
3. המעבר הזה הוא פשוט המסקנה שכתובה לעיל.
4. נובע מהגדרת m.

בשיעור הבא נראה איך להיעזר בטענה שהוכחנו כדי לחשב את LCPA ביעילות.

שיעור 11 – 28.12.2020

בעיית מנוע החיפוש, פתרון 2 – המשך:

בחלק הראשון של השיעור חזרנו על החומר של השיעור הקודם לשם הבהרה וריענון, תוכן החלק הזה של השיעור נמצא בסיכום השיעור הקודם.

כעת, נראה איך להיעזר בטענת מערך ה-LCP מהשיעור הקודם כדי לחשב את LCPA ביעילות:

האלגוריתם לבניית LCPA:

1. לכל i מ-1 עד n בצע:
   - חשב את $\mathrm{LCPA}\left[\mathrm{SA}^{-1}[\mathrm{i}]\right]$ החל מהתו $\max\left\{1, \mathrm{LCPA}\left[\mathrm{SA}^{-1}[\mathrm{i}-1]\right]-1\right\}$.

     לא צריך להשוות מהתו הראשון כי הוכחנו ש-$\mathrm{LCPA}\left[\mathrm{SA}^{-1}[\mathrm{i}-1]\right]-1$ התווים הראשונים של השתיים זהים.

     בנוסף, הוספנו את ה-max עם 1 לביטוי כי $\mathrm{LCPA}\left[\mathrm{SA}^{-1}[\mathrm{i}-1]\right]-1$ יכול להיות שווה 0 וזה לא מוגדר כי מחרוזת מתחילה מהתו ה-1.

2. החזר את LCPA.

זמן ריצה של בניית LCPA - $O(n)$:

לולאה עם n איטרציות, כך שסכום הפעולות בהן הוא כעלות סריקת הסיפא הארוכה ביותר, כלומר, $O(n)$. עם תיקוני התזוזות שמאלה, מכיוון שיש $O(n)$ כאלו, זמן הריצה הוא בכל מקרה לינארי.

כשמחשבים את ה-LCP של שתי מחרוזות כדי למלא תא ב-LCPA, מתחילים להשוות ביניהן בתו שהוא אחד לפני התו בו הפסקנו להשוות את השתיים הקודמות. אם היינו מתחילים את ההשוואה בדיוק איפה שהפסקנו בשתי המחרוזות הקודמות אז למעשה היינו מבצעים לאורך כל האיטרציות סריקה בודדת שלמה של מחרוזת באורך הסיפא הארוכה ביותר, ב-$O(n)$. למעשה, בכל השוואה בין שתי מחרוזות נצטרך להתקדם תו אחד לכל היותר כדי להגיע לתו בו הפסקנו להשוות את המחרוזות הקודמות, אז יש לנו רק "תיקון" אחד בכל איטרציה ובסך הכל יש לנו $O(n)$ "תיקונים" כדי להגיע למצב של סריקה בודדת בעלות $O(n)$. לכן, בסך הכל העלות היא $O(n)$.

לסיום הנושא, נחזור לאופן בניית המערך SA ב-$O(n)$:

לשם כך, נזכיר בקצרה מספר אלגוריתמים שנלמדים בקורס "מבני נתונים" (או "DAST"):

- **Merge Sort:**

נתון מערך A בגודל n ורוצים למיין אותו. מחלקים את המערך ל-2 חלקים בני $\frac{n}{2}$ איברים (החלוקה יכולה להיות למערך איברים עם אינדקס זוגי ומערך איברים עם אינדקס אי-זוגי) וממינים כל חלק בעלות $T\left(\frac{n}{2}\right)$ ואז מאחדים את שני החצאים הממוינים ב-$O(n)$. בסך הכל העלות היא $2 \cdot T\left(\frac{n}{2}\right) + O(n)$ ולפי משפט האב זה יוצא $\boldsymbol{O(n \cdot \log n)}$.

- **Counting Sort:**

מיון זה יעיל כשאנחנו יודעים את טווח הערכים האפשריים, נסמנו ב-k, והוא אינו גבוה מדי.

נסביר באמצעות דוגמה:

נניח שיש לנו את המערך $A = [1,7,5,1]$ ואנחנו רוצים למיין אותו.

נקצה מערך B בגודל 7 עם 0 בכל איבריו, נסרוק את A משמאל לימין ובכל פעם שנתקלים באיבר x כלשהו מעלים את $B[x]$ באחד. נקבל את המערך B הבא:

1	2	3	4	5	6	7
2	0	0	0	1	0	1

מ-B אפשר לבנות בקלות מערך ממוין המכיל 2 ערכי 1, 5 אחד ואז ערך 7 אחד: $[1,1,5,7]$.

זמן הריצה הוא זמן הריצה של מעברים בודדים על A ועל B ולכן העלות היא $\boldsymbol{O(n+k)}$.

- **Radix Sort:**

נניח שיש לנו את המערך $[23,27,72]$ ואנחנו רוצים למיין אותו.

Radix sort ממין מיון ראשוני לפי הספרה השמאלית ביותר, מיון שניוני לפי הספרה מימינה וכן הלאה. Radix Sort מבצע זאת כך: הוא ימיין קודם לפי הספרה הימנית ביותר (הכי פחות משמעותית) ואז לפי הבאה בתור עד לספרה השמאלית ביותר (ממינים גם לפי הספרות הכי ימניות לצורך "שובר שוויון" כמו במקרה של 23 ו-27). נשים לב שהמיון האחרון יהיה הכי דומיננטי והשאר יהוו שוברי שוויון.

כל מיון לפי ספרה מסוימת יעשה באמצעות Counting Sort, כאשר k הוא הבסיס בו אנחנו עובדים – במקרה שלנו 10.

המיון הראשון ימלא את מערך העזר הבא:

0	1	2	3	4	5	6	7	8	9
0	0	1	1	0	0	0	1	0	0

וממנו את מערך הביניים $[72, 23, 27]$.

המיון השני יבנה את מערך העזר הבא:

0	1	2	3	4	5	6	7	8	9
0	0	2	0	0	0	0	1	0	0

וממנו את המערך הממוין הסופי $[23, 27, 72]$.

אנחנו יודעים שהסדר לא השתבש בין המיונים כי counting sort הוא מיון יציב.

זמן ריצה: $O\left(\#digits\ in \max number \cdot (n+d)\right)$, כש-d הוא מספר הספרות בבסיס.

עבור בסיסים קבועים, $O(n+d) = O(n)$.

כמו כן, $\#digits\ in \max number = \log_d(max\ number)$.

<u>הגדרות:</u>

i. <u>מערך סיפות חלקי של T</u> - הוא תת-סדרה של מערך הסיפות של T.

דוגמה: עבור T=ABCDEFABCDFF מערך **מחרוזות** הסיפות הממוין, SA(T), יראה כך:

ABCDEFABCDFF	ABCDFF	BCDEFABCDFF	BCDFF	CDEFABCDFF	CDFF	DEFABCDFF	DFF	EFABCDFF	FABCDFF	FF	F

מערך הסיפות הממוין:

1	7	2	8	3	9	4	10	5	6	11	12

מערך **מחרוזות** סיפות חלקי של T הוא, למשל:

ABCDEFABCDFF	BCDFF	CDFF	DFF

ומערך סיפות חלקי מתאים יהיה:

1	8	9	10

ii. <u>מחרוזת בלוקים</u> - נחלק את T לבלוקים בגודל 3, כל שלשת אותיות תגדיר בלוק כזה.
מיון (הסיפות) של המחרוזת החדשה יגדיר, עם טרנספורמציה מתאימה עליו, מערך סיפות חלקי של T שמתאים לאינדקסים שהם $mod 3$ 1.

עבור הדוגמה שלנו נחלק את T לבלוקים הבאים:

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

נמיין את הסיפות של המחרוזת החדשה שקיבלנו, המורכבת מאותיות חדשות שיצרנו, כל אחת בת 3 אותיות מקוריות, ונקבל:

1	3	2	4

.

אם נפעיל את המיפוי $3(x-1)+1$ על כל אחד מאברי המערך לעיל, נקבל מערך סיפות חלקי מתאים של T:

1	7	4	10

iii. <u>מערך הסיפות המוכלל של המחרוזות A,B</u> - הוא מערך של כל (האינדקסים של) הסיפות של A ו-B ממוינות. לענייננו, נגדיר אותו בתור $SA(A\$B)$ (כלומר, מערך הסיפות של המחרוזת $A\$B$), כאשר $ הוא תו גדול מכל תו אחר.

<u>האלגוריתם:</u>

1. צור מ-T שתי מחרוזות B_1, B_2 שהן מחרוזות בלוקים בגודל 3.
B_1 – חלוקה של T לבלוקים בגודל 3.
B_2 – חלוקה של T לבלוקים בגודל 3 החל מהאינדקס השני ב-T.
עבור הדוגמה שלנו:

B_1:

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

B_2:

B	C	D	E	F	A	B	C	D	F	F	#

מציין תו מסיים מחרוזת.

2. הגדר שמות מילוניים לבלוקים שהוגדרו על ידי מיונם ב-$O(n)$.
נגדיר "שם מילוני" של אות כמספר המייצג כמה אותיות (מהסוג החדש שמכיל 3 אותיות רגילות) יש לנו שקטנות ממנה. בדוגמה שלנו יתקבלו השמות המילוניים הבאים:

אות:	A	B	C	D	E	F	A	B	C	D	F	F
שם מילוני:	0			4			0			5		

אות:	B	C	D	E	F	A	B	C	D	F	F	#
שם מילוני:	2			6			2			7		

כך שלמעשה ביצענו את שני התרגומים הבאים: $T_1 = 0405 \leftarrow B_1$ וגם $T_2 = 2627 \leftarrow B_2$.
כתוצאה מבחירת השמות המילוניים, האפשרויות שיש לנו לספרה נתונה נעות בין 0 ל-n, שכן כל ספרה מציינת כמה איברים קטנים מהאות שלה יש, ויש לא יותר מ-n אותיות.

3. קרא רקורסיבית לאלגוריתם על המחרוזת $U = T_1\$T_2$ (כלומר, צור מערך סיפות מוכלל עבור אינדקסים שהם שונים מ-$0\ mod 3$ עד כדי התרגום כפי שהגדרנו בהגדרת מחרוזות בלוקים).
קיבלנו מערך סיפות מוכלל של U.

4. תרגם את מערך הסיפות המוכלל $SA(U) = S'_{1,2}$ שהם האינדקסים של חלקי סיפות למערך שונים מ-$3\ mod\ 0$ של T:

$$S_{1,2}[i] = \begin{cases} 3\left(S'_{1,2}[i]-1\right)+1 & S'_{1,2}[i] \in [1,\dots,|T_1|] \\ 3\left(S'_{1,2}[i]-(|T_1|+1)-1\right)+2 & S'_{1,2}[i] \in [|T_1|+2,\dots,|T_1|+|T_2|+1] \end{cases}$$

אם $S'_{1,2}[i] \in [1,\dots,|T_1|]$, אז אנחנו יודעים שהסיפא הזאת באה מהחלק של $T_1\$T_2$ שקודם לסימן הדולר ולכן היא מתאימה לסיפא שמתחילה באינדקס שהוא $3\ mod1$ של T.

אם $S'_{1,2}[i] \in [|T_1|+2,\dots,|T_1|+|T_2|+1]$, אז אנחנו יודעים שהסיפא הזאת באה מהחלק של $T_1\$T_2$ שהוא אחרי סימן הדולר ולכן היא מתאימה לסיפא שמתחילה באינדקס שהוא $3\ mod2$ של T.

בשיעור הבא נטפל בסיפות של T שמתחילות באינדקסים שהם $0\ mod3$ ובכך נסיים את בניית SA(T).

שיעור 12 – 04.01.2020

בעיית מנוע החיפוש – המשך:

עד עכשיו, עברנו על 4 שלבים של האלגוריתם לבניית המערך SA ב-$O(n)$ וקיבלנו את $S_{1,2}$, מערך הסיפות החלקי של אינדקסים שאינם $0\ mod3$ (כלומר, קיבלנו את כל הסיפות המתחילות מאינדקסים שאינם $0\ mod3$, ממוינות). כעת נראה איך למיין את הסיפות של T שמתחילות באינדקסים שהם $0\ mod3$ ולשלב אותן עם שאר הסיפות:

5. **צור מיפוי הפוך ל-$S_{1,2}$** – כלומר, מיפוי בעזרתו נדע עבור כל אינדקס שלא מתחלק ב-3 מה מקום הסיפא שמתחילה ממנו במיון: נגדיר מערך בגודל T, בתא הראשון שלו יהיה המיקום במיון של הסיפא שמתחילה מ-$T[1]$, בתא השני יהיה המיקום במיון של הסיפא שמתחילה מ-$T[2]$ ובתא השלישי נשים תו ייחודי כלשהו, למשל *, שכן האינדקס כן מתחלק ב-3 ועוד לא הכנסנו אותו למיון. ככה נמשיך עד סוף המערך החדש.

6. **צור את מערך הסיפות החלקי של T של אינדקסים המתחלקים ב-3** (נסמן מערך זה ב-C) – בעזרת שתי הרצות של מיון בסיס באמצעות התייחסות לכל סיפא כמספר דו-ספרתי עם ספרות בין 0 ל-n: הספרה הראשונה היא האות הראשונה בסיפא והשנייה היא מיקומה של הסיפא שמתחילה באינדקס העוקב (כלומר, שונה מ- $0\ mod3$) לפי המיון.

למשל, עבור הדוגמה מהשיעור הקודם נקבל את מערך הסיפות החלקי הבא עבור אינדקסים שמתחלקים ב-3:

C	D	E	F	A	B	C	D	F	F	#	#

נתייחס לבלוקים כמספרים דו-ספרתיים באופן הבא:

מספר בלוק	ספרה ימנית	ספרה שמאלית
1	מיקום DEFABCDFF ב-$S_{1,2}$ = 5	C
2	המיקום של ABCDFF ב-$S_{1,2}$ = 2	F
3	מיקום DFF ב-$S_{1,2}$ = 6	C
4	*	F

7. **מיזוג המערכים:**

נזכור כי כשמשלבים את שני המערכים משווים בכל שלב סיפא מ-C וסיפא מ-$S_{1,2}$.

א. <u>אם הסיפא מ-$S_{1,2}$ היא $1\ mod3$</u>, נשווה את האותיות הראשונות ובתור שובר שוויון נשתמש במיקום במיון של הסיפות שמתחילות באינדקס העוקב של שתיהן.

ב. <u>אם הסיפא מ-$S_{1,2}$ היא $2\ mod3$</u>, משווים שתי אותיות ראשונות וכשובר שוויון לוקחים את המיקום במיון של הסיפות שמתחילות שני אינדקסים אחרי תחילת כל אחת מהן.

למה עושים את זה, ולא משתמשים כשובר שוויון בסיפות שמתחילות באינדקס העוקב לכל אחת, כמו בסעיף א'? כי אם יש שוויון באות הראשונה של שתי הסיפות, האינדקס העוקב לסיפא מ-$S_{1,2}$ מתחלק ב-3 ועדיין לא השווינו בין $S_{1,2}$ ל-C! אם נתקדם שני אינדקסים, נגיע לסיפא שמתחילה באינדקס שהוא $1\ mod3$ וסיפא שמתחילה באינדקס שהוא $2\ mod3$ ואת שתיהן כבר מיינו באותו המערך.

8. החזר את המערך המאוחד.

<u>**זמן ריצה - $O(n)$:**</u>

1. **חלוקה לבלוקים - $O(n)$**.
2. **הגדרת שמות מילוניים – $O(n)$:** תעשה באמצעות מיון בסיס (כל בלוק בנוי משלוש אותיות מקוריות, לכן אפשר למיין את כל הבלוקים באמצעות שלוש הרצות של counting sort) **בעלות** $O(n)$.
3. **קריאה רקורסיבית על $T_1\$T_2$ – $O(n)$:** יצירת $T_1\$T_2$ עולה $O(n)$, גודלה $\frac{2n}{3}+1$ ולכן עלות הקריאה הרקורסיבית עליה היא $T\left(\frac{2n}{3}+1\right)$.
4. **מיפוי - $O(n)$**.
5. **מיפוי הפוך - $O(n)$:** סורקים את המיפוי ואם באינדקס ה-i שלו רואים את האיבר ה-j במערך המקורי, כותבים בתא ה-j של המיפוי ההפוך i.
6. **יצירת מערך הסיפות החלקי שמתאים לאינדקסים המתחלקים ב-3 - $O(n)$:** מיון בסיס על דו-ספרתיים.
7. **מיזוג המערכים - $O(n)$:** במקרה א' משווים אות ומקום של סיפא במיון ובמקרה ב' משווים שתי אותיות ומקום של סיפא במיון – בכל מקרה השוואה ב-$O(1)$.
8. **החזרת המערך המאוחד - $O(n)$**.

בסך הכל - העלות היא $T\left(\frac{2n}{3}+1\right)+O(n)$ ולפי משפט האב, בהנחה שמתקיים $\mathrm{T}(1)=1$, זה שווה ל-$O(n)$.

פתרון הבונוס של שאלה 1, תרגיל 3:

בונוס: לאחר שיפור התשתיות, הוחלפה מערכת הגשרים במערכת עמידה יותר. בכל עיר הוקמה נקודת בקרה האחראית על כל הגשרים היוצאים מהעיר, וכדי לפוצץ גשר יש לפוצץ את שתי נקודות הבקרה שבשני קצוותיו.

שניים מהחברים במחתרת סיכמו על סידור של הגשרים. הם מתחרים בתורות לסירוגין, כך שבתור ה־k, המתמודד שזהו תורו יכול לפוצץ את אחת מנקודות הבקרה של הגשר ה־k לפי סידור זה. אם בתור זה, שתי הנקודות כבר פוצצו, השחקן שזהו תורו מפסיד. אם כעבור $n-1$ תורות אף מתמודד לא הפסיד, מוכרז שוויון.

הניחו כי שני השחקנים משחקים ללא טעויות, כך שעבור סידור נתון, אם לאחד השחקנים יש אסטרטגיה מנצחת הוא ישאף לנצח בתור המוקדם ביותר ויריבו ינסה לדחות את הפסדו ככל שיוכל. תארו אלגוריתם אשר בהינתן העץ וסידור הגשרים בו, מחזיר את התור בו המשחק מסתיים. פתרונות שזמן ריצתם הוא $2^{O(n)}$ יזכו ב־10 נקודות בונוס ופתרונות שזמן ריצתם הוא $O(n^2)$ יזכו ב־40 נקודות. הסבירו נכונות וזמן ריצה.

קלט: עץ עם סידור הצלעות.

פלט: התור בו המשחק מסתיים.

בכל תור i השחקן שזה תורו בוחר קודקוד מהצלע ה-i. אם הוא לא יכול (כלומר, שני קודקודי הצלע כבר נבחרו) הוא מפסיד. שני השחקנים משחקים אופטימלית.

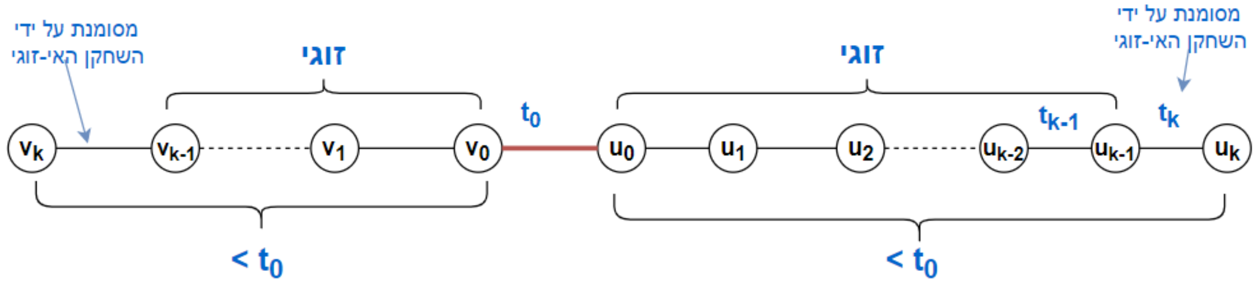
באופן כללי, יכולנו להשתמש בשיטה שנקראת "אינדוקציה לאחור" – זו שיטה שתעבוד עבורנו אך היא לא תמיד מאוד יעילה. במקרה שלנו היא תניב תוצאה לא יעילה מספיק, כזו שמתאימה לבונוס הקל יותר. השתמשנו בדוגמה של משחק מרבה הרגליים כדי להראות איך לבצע אינדוקציה לאחור.

הגדרה – צלע מכריעה:

צלע $e=\{u_0,v_0\}$ תקרא מכריעה אם קיימים מסלולים $v_0 \neq u_1,\dots,u_k$ ו-$u_0 \neq v_1,\dots v_{k'}$ (לא בהכרח מתקיים $k=k'$) העונים לתנאים הבאים:

נניח שהצלע t_0 מתאימה לשחקן הזוגי (זה שמשחק באיטרציות שמספרן זוגי) (האי-זוגי):

- t_k מתאימה לשחקן האי-זוגי (הזוגי).
- המספר של כל הצלעות בשני המסלולים קטן מזה של t_0.
- t_{k-1} תהיה מאוחרת מ-t_k.
- כל הצלעות שבין u_0 ל-u_{k-1} מתאימות לשחקן הזוגי (האי-זוגי).
- הצלע שבין v_k ל-v_{k-1} מתאימה לשחקן האי-זוגי (הזוגי).
- כל הצלעות שבין v_0 ל-v_{k-1} מתאימות לשחקן הזוגי (האי-זוגי).



<u>טענה:</u>

א. אם e היא צלע מכריעה, אז התור t_0 (מספר התור בו משחקים את הצלע ה-e), או תור מוקדם ממנו, הוא התור בו המשחק מסתיים.

ב. אם המשחק מסתיים בתור t_0 אז e היא צלע מכריעה.

<u>הוכחה:</u>

א. נניח כי e מכריעה:

ניתן לשחקן האי-זוגי לבחור את הנקודה u_{k-1}.

אם u_{k-1} כבר נבחרה הוא יבחר את u_k.

אם גם u_k נבחרה, אז התור בו המשחק הסתיים מוקדם מהתור ה-t_0, כנטען.

לאחר מכן, נותרו לכל היותר k-1 קודקודים המתאימים ל-k-1 תורות של השחקן הזוגי, לכן בכל התורות האלו יבחר אותם – אם לא יכול, יש תור מכריע מוקדם לתור ה-t_0 וזה תואם את הטענה.

לכן, בתור ה-t_0 שאחריהם, השחקן הזוגי מפסיד כי u_0 ו-v_0 נבחרו.

נראה כי הבחירה האסטרטגית של השחקן האי-זוגי, לסמן את הנקודה ה-u_{k-1} או ה-u_k, חוקית:

ננתח מתי הבחירה הזו יכולה להיות לא חוקית, כלומר, לגרום לשחקן האי-זוגי להפסיד דרך מסלול או תת-עץ אחר אשר יוצא מהקודקוד u_{k-1} ומחובר אליה באמצעות צלע שנסמנה t':

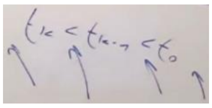
1. <u>t' היא צלע של השחקן הזוגי:</u> במקרה הזה השחקן האי-זוגי בסך הכל מכריח את השחקן הזוגי לבחור את הקודקוד השני של t'. השחקן הזוגי יכול היה לבחור את הקודקוד הזה בלי שום קשר למה שהשחקן האי-זוגי בחר ולכן אותה שרשרת אירועים שעלולה הייתה לגרום לשחקן האי-זוגי להפסיד בעקבות הבחירה ב-u_{k-1} עדיין יכולה לקרות ולכן לא תלויה בבחירה ב-u_{k-1}. כלומר, במקרה הזה הבחירה ב-u_{k-1} חוקית.

2. <u>t' שייכת לשחקן האי-זוגי אך מקיימת $t_0 < t'$:</u> המקרה הזה לא מפריע לנו כי גם אם השחקן האי-זוגי יכול היה איכשהו להפסיד בגלל t', המשחק כבר מסתיים לפני כן, בתור ה-t_0. כלומר, במקרה הזה הבחירה ב-u_{k-1} חוקית.

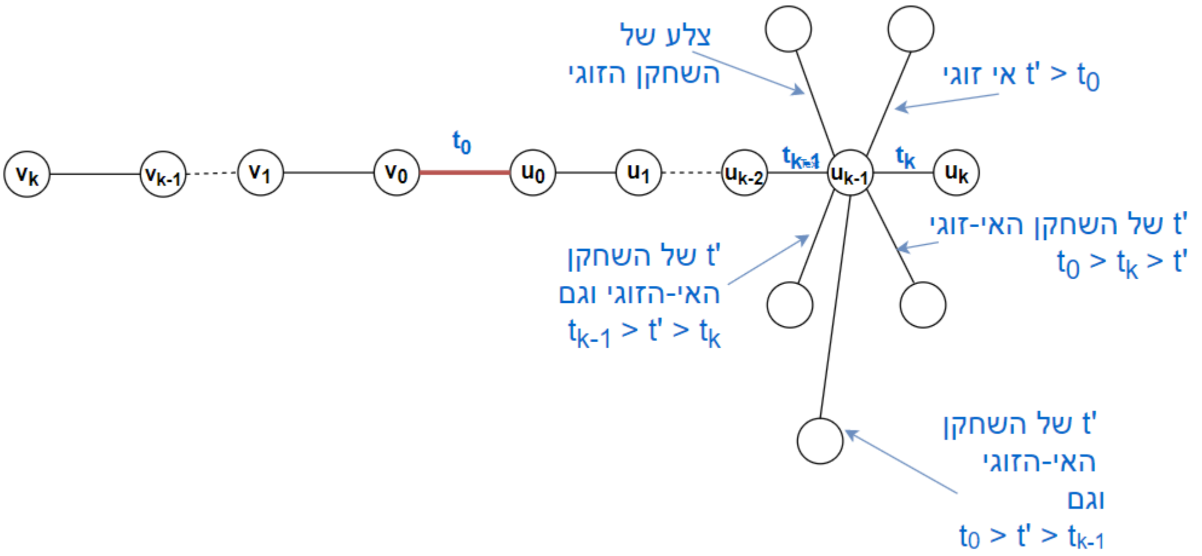
3. <u>t' שייכת לשחקן האי-זוגי ומתקיים $t' < t_k < t_0$ (t_k היא הצלע שבין u_k ל-u_{k-1}):</u> הבחירה בקודקוד השני של t' נעשתה לפני הבחירה ב-u_{k-1} והבחירה לסמן את u_{k-1} לא תשפיע בכלל על תת העץ המושרש ב-u_{k-1} ויוצא דרך t'. כלומר, הבחירה ב-u_{k-1} חוקית.

4. <u>t' שייכת לשחקן האי-זוגי, אינה גדולה מ-t_0 ומתקיים $t_k < t' < t_{k-1}$:</u>
במקרה הזה, נסמן בניתוח שלנו את t' בתור t_k **במקום** t_k המקורית ולכן המקרה לא רלוונטי.

5. <u>t' שייכת לשחקן האי-זוגי ומתקיים $t_{k-1} < t' < t_0$:</u> התת-עץ הזה יכול להשתבש, אבל בגלל שמתקיים $t' < t_{k-1}$ וגם t_{k-1} היא צלע של השחקן השני, אם השחקן האי-זוגי לא היה בוחר את u_{k-1} אז השחקן הזוגי היה בוחר את u_{k-1}. לכן הבחירה של השחקן האי זוגי ב-u_{k-1} לא משפיעה על ההפסד דרך התת-עץ הזה.



ההוכחה לגבי המסלול שקודקודיו מסומנים עם האות v סימטרית.

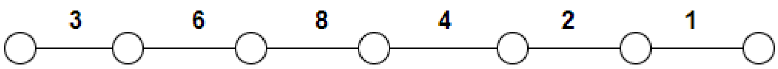


בשיעור הבא נוכיח את סעיף ב' של הטענה.

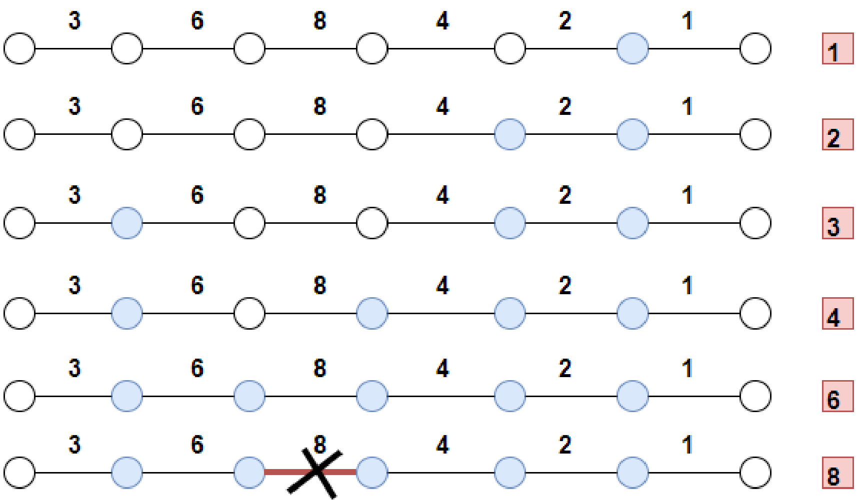
שיעור 13 – 12.01.2020

פתרון הבונוס של שאלה 1, תרגיל 3 - המשך:

להלן דוגמא לשני מסלולים שיוצאים מצלע מכריעה, כפי שתיארנו בשיעור הקודם:



מהרגע שהשחקן האי-זוגי יסמן את הקודקוד השמאלי של צלע 1, הוא יתניע סדרת בחירות שתכפה על השחקן הזוגי, כך שבתור מספר 8 השחקן הזוגי יפסיד:



(מימין – מספר התור)

נזכר בטענה שהתחלנו להוכיח בשיעור הקודם:

טענה:

א. אם e היא צלע מכריעה, אז התור t_0 (מספר התור בו משחקים את הצלע ה-e), או תור מוקדם ממנו, הוא התור בו המשחק מסתיים.

ב. אם המשחק מסתיים בתור t_0 אז e היא צלע מכריעה.

המשך הוכחה:

השלמה להוכחת סעיף א' של הטענה מהשיעור הקודם:

הבחירה בקודקוד המדובר (למשל הקודקוד השמאלי של צלע מספר 1 בדוגמה לעיל) אולי אינה אופטימלית עבור השחקן שבחר אותו (כלומר, אולי עדיף מבחינתו לבחור את הקודקוד השני של הצלע), אך אם אינה אופטימלית המשחק יסתיים בתור מוקדם יותר וזה עדיין מוכיח את הטענה.

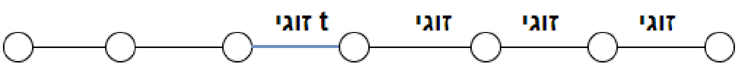
נוכיח את סעיף ב' של הטענה: נניח, בלי הגבלת הכלליות, ש-t זוגי.

נראה, בעזרת אינדוקציה על אורך המסלול, כי אם t הוא תור מכריע, אז הוא מתאים לצלע מכריעה.

נראה כי אם אין לצלע המתאימה ל-t מסלול ימני או מסלול שמאלי כמבוקש, אז t אינו תור מכריע:

- **אם במסלול שיוצא מהצלע t כל הקשתות זוגיות**, אז השחקן בתור ה-t יוכל לבחור קודקוד מצלע זו כי אין דרך לאכוף שלא יבחר כזה.

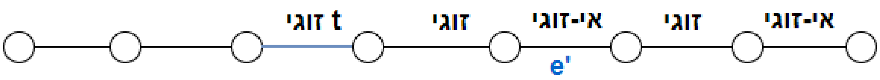
  למשל, בדוגמה הנ"ל, השחקן הזוגי יוכל לבחור את הקודקודים הימניים ביותר של כל הצלעות במסלול הימני היוצא מ-t, כך שכשיגיע התור t בוודאות הקודקוד הימני של הצלע t עדיין לא ייבחר ולכן השחקן הזוגי לא יפסיד בתור זה.



- **אם במסלול יש יותר מצלע אחת אי-זוגית**, אז:

  עבור הרישא שלה שכוללת צלע אי-זוגית יחידה, e': אינדוקטיבית לא ניתן לאכוף ניצחון (כלומר, משום שאנחנו מוכיחים באינדוקציה על אורך המסלול אנחנו מניחים שכבר הראינו בשלב קודם של האינדוקציה כי אי אפשר לאכוף הפסד של השחקן הזוגי מהרישא של המסלול).

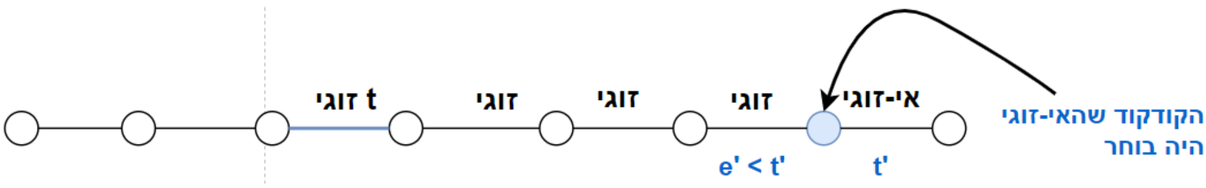
  עבור ההמשך שלה: הוא לא משפיע כי הפעפוע של ההשפעה הוא עד הצלע e' (השחקן האי-זוגי יתניע מהלך מקצה המסלול, אך כאשר יגיע התור e' זו כבר שוב בחירה שלו, כלומר, השפעת הבחירה הראשונה של השחקן האי-זוגי על השחקן הזוגי נפסקת בצלע e').



- **אם יש צלע במסלול עם תור גדול מ-t**, אז:

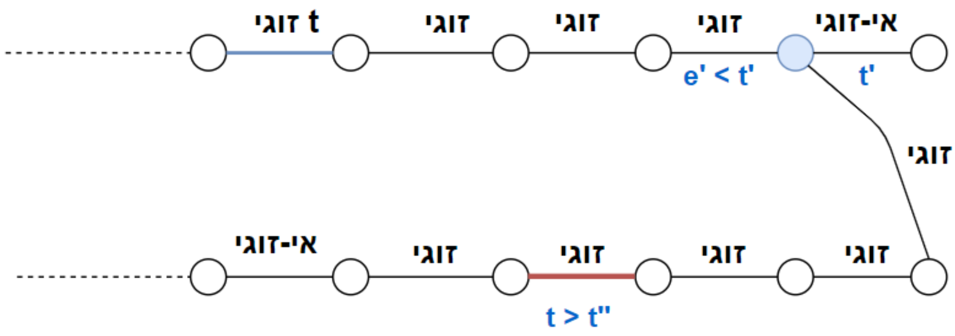
  הרישא של המסלול הזה שלא כוללת את הצלע: אינדוקטיבית לא יכולה לגרום לניצחון.

  ההמשך של המסלול: לא יכול לאכוף ניצחון כי הוא לא מפעפע עד ל-t (סדרת האילוצים על השחקן הזוגי אותה מפעילה הבחירה הראשונה במסלול של השחקן האי-זוגי נעצרת ותמשך רק אחרי שהצלע t תבחר).



- **אם יש רצף צלעות זוגיות שמסתיים בצלע אי זוגית שהיא גדולה מקודמתה**, השחקן הזוגי יוכל לבחור את הקודקוד שהשחקן האי-זוגי היה בוחר וכך, בתור t, יוכל לבחור קודקוד כלשהו.

הבחירה של השחקן הזוגי בקודקוד זה היא לגיטימית. אם היא לא לגיטימית, אז, אינדוקטיבית, מכיוון ש-$t > t''$ (t'' היא הצלע שבגללה הזוגי יפסיד מוקדם מהתור ה-t), אז מימין ומשמאל ל-t'' יש רצף של צלעות זוגיות קטנות מ-t'' ובאחד הכיוונים צלע אי-זוגית שקטנה מהקודמת לה. לכן, יתקבל מסלול מהצלע t שמקיים את התנאי שהנחנו שלא מתקיים וזו סתירה.



אנחנו חוששים ממצב בו הבחירה של הקודקוד הכחול על ידי השחקן הזוגי תתניע תהליך שיגרום לשחקן הזוגי להפסיד בתור ה-t'', המוקדם יותר מהתור ה-t.

<u>האלגוריתם:</u>

1. לכל צלע e בעץ:
   א. הסר את e מהעץ.
   ב. הרץ DFS משני הקודקודים של e ובדוק האם קיימים שני מסלולים העונים לאפיון.
   ג. אם קיימים מסלולים כאלו – מסמנים את e.

2. החזר את הצלע המינימלית מתוך כל הצלעות שסומנו.

<u>נכונות:</u> נובעת מהטענה.

<u>זמן ריצה:</u>

1. לולאה חיצונית ב-$O(|E|)$ ופנימית בעלות DFS בעץ שהתקבל, כלומר $O(|V| + |E|)$ – בסך בכל $O(n^2)$.
2. בעלות של $O(n)$.

בסך הכל, עלות האלגוריתם היא $O(n^2)$, כנדרש.

רשימת דילוגים

מילון (Dictionary \ Map \ Associative Array) - הוא מבנה נתונים אבסטרקטי (כלומר, ניתן למימוש על ידי מבנה נתונים קונקרטי) שתומך בחיפוש, הוספה, הסרה, מציאת קודם ועוקב.
כלומר, באופן כללי, המבנה מאפשר גישה לערכים על ידי מפתחות, כך שלמפתח נתון קיים ערך יחיד (בשונה ממערך רגיל, בו מגדירים את המפתחות להיות n,…,1).

<u>הערות</u>

1. נציג מבנה נתונים הסתברותי במובן שהוא תמיד מחזיר תוצאה נכונה, אך זמן הריצה שלו יהיה בתוחלת (כי הוא לא דטרמיניסטי). כמסקנה מעובדה זו, ייתכנו סדרות שונות של הרצות של פעולות שיניבו מבנים שונים המייצגים את אותה הסדרה.

2. מימוש מוכר למילון הוא עץ חיפוש מאוזן (למשל AVL). מבנה הנתונים שנציג לעומתו הוא הסתברותי ולו יתרונות אחרים על פני עץ זה (היכולת לבצע מספר פעולות בו זמנית, קלות מימוש, היכולת לעבור מהערך של מפתח לערך של המפתח הבא או הקודם). בניגוד לטבלת גיבוב, המבנה שנציג תומך בפעולות קודם / עוקב בצורה יעילה.

3. ההסתברות אינה על גבי התפלגות ממנה נלקחים הקלטים, אלא על פני הביטים הרנדומיים שמוגרלים בפעולות (כלומר, המשתנה המקרי שלנו הוא לא הקלט של האלגוריתם, אלא ההגרלות שהוא מבצע ולכן תוחלת זמן הריצה לא תלויה בהתפלגות הקלטים).

<u>הדרישה מהמילון:</u>

1. <u>חיפוש(key):</u>

   החזר איטרטור לרשומה $(\widetilde{key},\widetilde{value})$ המקיימת כי $\widetilde{key}$ הוא המקסימלי שאינו גדול מ-key.

2. <u>הוספה$(key, value)$:</u>

   הוסף את הרשומה $(key, value)$ למבנה, אם היא לא קיימת, והחזר איטרטור לה.

3. <u>הסרה(key):</u>

   הסר את הרשומה עם המפתח key מהמבנה.

4. <u>עוקב$(iterator)$:</u>

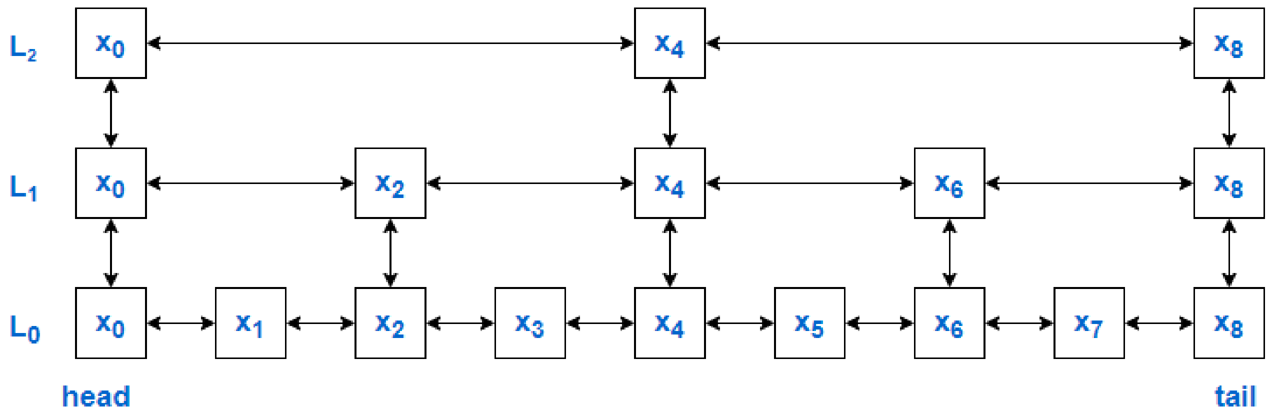
   החזר איטרטור לרשומה העוקבת לזו ש-$iterator$ מצביע עליה.

5. <u>קודם $(iterator)$:</u>

החזר איטרטור לרשומה הקודמת לזו ש-$iterator$ מצביע עליה.

<u>רשימת דילוגים דטרמיניסטית:</u>

דוגמה:



<u>תכונות המבנה:</u>

1. כל איברי המבנה נמצאים ברשימה התחתונה ביותר, שנקרא לה L_0.
2. כל האיברים ברשימה L_i נמצאים גם בזו שמתחתיה.
   - ב-L_0 נמצאים $x_0, x_1, \dots, x_n$.
   - ב-L_1 נמצאים $x_0, x_2, x_4, \dots, x_n$.
   - ...
   - ב-L_k (עבור $k = O(\log n)$) נמצאים $x_0, x_{\frac{n}{2}}, x_n$.

<u>מימוש הפעולות:</u>

- <u>פעולת החיפוש:</u>

מתחילים מהרשימה העליונה ביותר ומחפשים איבר מקסימלי שאינו גדול מהמפתח.

כשמוצאים, עוברים לרשימה מתחת ומתקדמים באופן דומה – כך עד L_0.

<u>זמן ריצה</u>: מעבר על $O(\log n)$ רשימות עם מספר קבוע, כלומר $O(1)$, נסרקים בכל רשימה.

הערה: מבנה זה לא תומך בהוספה או הסרה יעילה.

רשימת דילוגים הסתברותית:

תכונות המבנה:

1. הרשימה L_0 מכילה את כל האיברים.
2. כל האיברים ב-L_i נמצאים גם ב-L_{i-1}.
3. לכל איבר x, מוגדר גובהו, $h(x)$, שהוא מספר הרשימות בהן הוא קיים: $L_0, L_1, \dots, L_{h(x)-1}$ אך הוא לא קיים ב-$L_{h(x)}$.

מימוש הפעולות:

1. חיפוש(key):

   זהה לחיפוש בדטרמיניסטית, למעט כך שמתחילים מהרשימה L_M כאשר $M = \log_{\frac{1}{1-p}} n$.

   p מסמן את הטלות המטבע שהאלגוריתם מבצע.

בשיעור הבא נמשיך לתאר את מימוש הפעולות ברשימת דילוגים הסתברותית.

שיעור 14 – 12.01.2020

רשימת דילוגים הסתברותית – המשך

תכונות המבנה – המשך:

4. ברשימה יש איברי זקיף head (השמאלי ביותר) ו-tail (הימני ביותר) שלא משמשים לאחסון איברים וגובהם **לפחות** כגובה המקסימלי של איבר במבנה.

נניח שאנחנו מוסיפים למבנה איבר חדש שהגובה שלו יהיה יותר גבוה מכל האיברים הקיימים, אז נוסיף לרמה החדשה שלו גם את שני הזקיפים, כך שהגובה שלהם עדיין יהיה כמו האיבר הכי גבוה. אם נסיר את אותו איבר, אנחנו לא רוצים להקטין את גובה הזקיפים ולכן נגדיר את גובהם להיות לפחות כמו הגובה המקסימלי של איבר.

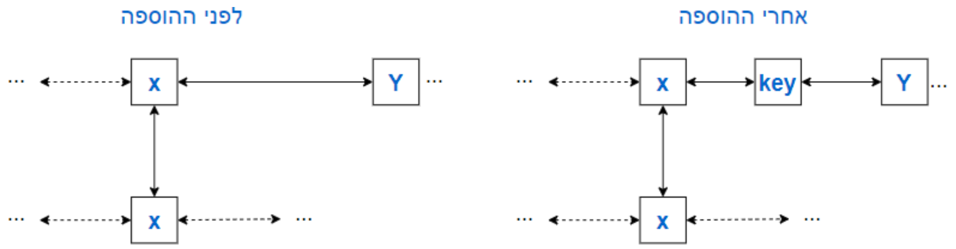
מימוש הפעולות – המשך:

1. חיפוש – הבהרה:

דיברנו כבר בשיעור הקודם על חיפוש, נחדד כי מחזירים איטרטור לאיבר ולא את האיבר עצמו. כלומר, כשנגיע לאיבר, אפילו אם ברמה גבוהה יחסית, נרד דרכו עד לרמה L_0 ונחזיר איטרטור.

2. הוספה(key, val):

מחפשים את key ברשימה, מגרילים עבורו גובה, $h(key) \sim Geo(p)$ (p הוא קבוע, בחרנו אותו עבור המבנה). בכל רמה בה נוסיף את key ניצור קישור לאיברים קודמים באמצעות הרצת חיפוש נוספת. באשר לעוקבים – הם פשוט העוקבים של הקודמים. כלומר, לאחר שנמצא את המקום בו נוסיף את key ב-L_0, נריץ שוב חיפוש. בכל רמה בחיפוש, כשנגיע ל-x, האיבר ממנו נרד לרמה הבאה, נוסיף את key לרמה עם x כקודמו ומי שהיה העוקב של x כעוקבו. למעשה, מוסיפים את key בכל רמה רלוונטית מלמעלה למטה.



3. הסרה(key):

חיפוש key ברשימה, קישור עוקבו וקודמו בכל רמה ולהפך והסרת key.

4. עוקב$(iterator)$:

החזר איטרטור המצביע למי שמימין ל-$iterator$ ברשימה L_0.

5. קודם$(iterator)$:

החזר איטרטור המצביע למי שמשמאל ל-$iterator$ ברשימה L_0.

<u>הנחה</u>: האיטרטורים הם לרשימה L_0.

<u>תוחלת המקום שדרוש למבנה אחרי n הוספות - $O(n)$:</u>

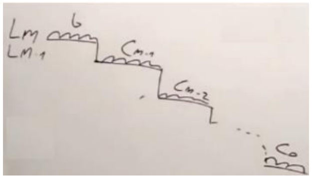
כלומר, n מסמן את כמות כל האיברים שהוכנסו עד כה (גם אם חלקם הוצאו) ולא כמה איברים יש עכשיו במבנה.

$$\mathbb{E}\left(\begin{matrix}\text{גודל}\\ \text{המבנה}\end{matrix}\right) = \mathbb{E}\left(\begin{matrix}\text{גובה}\\ \text{הזקיפים}\end{matrix} + \sum_{i=1}^{n} h(x_i)\right) \leq \mathbb{E}\left(3 \cdot \sum_{i=1}^{n} h(x_i)\right) = 3 \cdot \sum_{i=1}^{n} \mathbb{E}\big(h(x_i)\big) = 3 \cdot \frac{n}{p}$$
$$= \boldsymbol{O(n)}$$

<u>הסברים למעברים:</u>

1. הגובה של כל איבר הוא למעשה כמות המופעים שלו.
2. נובע משילוב שתי עובדות:
   א. גובה זקיף = הגובה המקסימלי של איבר $\leq$ סכום גבהי כל האיברים.
   ב. מונוטוניות התוחלת.
3. מלינאריות התוחלת.
4. הגבהים הם משתנים גיאומטריים (תוחלת כל אחד היא $\frac{1}{p}$).
5. p קבוע.

<u>תוחלת זמן הריצה של פעולת חיפוש - $O(\log n)$:</u>



נניח כי ברשימה L_M נסרקים b איברים וברשימה L_i לכל $0 \leq i \leq M-1$ נסרקים C_i איברים.

<u>ננתח את הגודל של b בתוחלת:</u>

$$\mathbb{E}(b) \leq \mathbb{E}(|L_M|) = 2 + \sum_{i=1}^{n} P(h(x_i) \geq M) = 2 + n \cdot (1-p)^M = 2 + n \cdot \frac{1}{\frac{1}{(1-p)^M}}$$

$$= 2 + n \cdot \frac{1}{\left(\frac{1}{1-p}\right)^M} = 2 + n \cdot \frac{1}{\left(\frac{1}{1-p}\right)^{\log_{\frac{1}{1-p}} n}} = 2 + n \cdot \frac{1}{n} = 3$$

<u>הסברים למעברים</u>

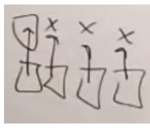
1. נובע משתי עובדות:
   א. גודל השורה L_M גדול שווה מ-b.
   ב. מונוטוניות התוחלת.

כמו כן, את ה-2 הוספנו כי 2 הזקיפים בוודאות יהיו ברשימה.

3. $|L_M| \sim Bin(n,(1-p)^M)$ כי אנחנו מבצעים n ניסויים, כאשר ההסתברות להצליח בכל ניסוי (כלומר להגיע עד לרשימה L_M היא $(1-p)^M$.
6. לפי הגדרת M.
7. נזכר בחוק $k^{\log_k n}$ מחוקי לוגים. אם נסמן $k=\frac{1}{1-p}$ נווכח כי מתקיים $\left(\frac{1}{1-p}\right)^{\log_{\frac{1}{1-p}} n}=n$.

<u>ננתח את הגודל של C_i בתוחלת:</u>

הבחנה: $C_i=1 \Leftrightarrow$ האיבר האחרון שנסרק ב-L_i הופיע גם ב-L_{i+1} (אם היה עוד איבר שנסרק ב-L_i ונמצא גם ב-L_{i+1}, אותו איבר היה נסרק ב-L_{i+1} והיינו יורדים **ממנו** אל L_i). באופן כללי, $C_i=k \Leftrightarrow k-1$ האיברים האחרונים שנסרקו ב-L_i לא מופיעים ב-L_{i+1} אבל האיבר ה-k שמשמאלם כן מופיע ב-L_{i+1}.



כמו כן, בשל תכונת חוסר הזיכרון של משתנה גיאומטרי, ההסתברות שאיבר x מופיע ברמה L_i אבל לא ברמה L_{i+1} היא פשוט ההסתברות שבהטלת המטבע הנוגעת לרמה L_{i+1} יצא "עץ", כלומר:

$$P\left(\text{לא מופיע } x \text{ ב} - L_{i+1} \,\middle|\, \text{מופיע } x \text{ ב} - L_i\right)=p \Rightarrow P\left(\text{מופיע } x \text{ ב} - L_{i+1} \,\middle|\, \text{מופיע } x \text{ ב} - L_i\right)=1-p$$

לכן, עבור ערכי k קטנים:

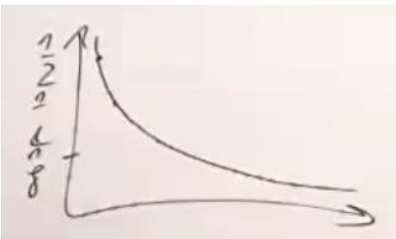
$$P(C_i=k)=p^{k-1}\cdot(1-p) \Rightarrow C_i \sim Geo(1-p)$$

מדוע עבור ערכי K קטנים?

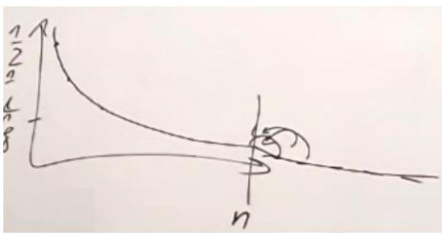
משום שיש אינפורמציה נוספת: יודעים שהזקיף בוודאות יופיע גם ב-L_{i+1} וגם ב-L_i.

כלומר, הניתוח של $p^{k-1}\cdot(1-p)$ נכון עבור ערכי k קטנים (לא עד הזקיף), לכן C_i כמעט גיאומטרי או "גיאומטרי עם זנב קטום":

עבור $p=\frac{1}{2}$ הגרף של משתנה גיאומטרי באמת יראה כך:



אך עבור משתנה גיאומטרי עם זנב קטום, בנקודה מסוימת הגרף נפסק ואת כל המשתנים ההסתברותיים מימין לשלב הזה מעבירים אל הנקודה הזו.



לכן, נקבל כי $\mathbb{E}(\mathrm{C_i})$ קטנה מתוחלת של $Geo(1-p)$.

<u>עכשיו נחבר את כל החלקים:</u>

$$\mathbb{E}\begin{pmatrix}\text{זמן ריצת}\\\text{החיפוש}\end{pmatrix} = \mathbb{E}\left(\mathrm{b} + \sum_{\mathrm{i=0}}^{\mathrm{M-1}} \mathrm{C_i}\right) = \mathbb{E}(\mathrm{b}) + \sum_{\mathrm{i=0}}^{\mathrm{M-1}} \mathbb{E}(\mathrm{C_i}) \le 3 + \mathrm{M} \cdot \frac{1}{1-p}$$

$$= 3 + \log_{\frac{1}{1-\mathrm{p}}} n \cdot \frac{1}{1-p} = \boldsymbol{O(\log n)}$$

<u>הסברים למעברים:</u>

2. מלינאריות התוחלת.
3. מהגדרת M.
4. 3 הוא קבוע, Log של n עם בסיס קבוע זה בערך $\log n$ ו-$\frac{1}{1-p}$ זה קבוע.

<u>תוחלת זמן הריצה של פעולת הוספה - $\boldsymbol{O(\log n)}$:</u>

עלות שני חיפושים $(O(\log n))$ ועוד הוספת איבר בגובה $\mathbb{E}(h(x)) = \frac{1}{p}$ ולכן זמן הריצה הוא $O\left(2\log n + \frac{1}{p}\right) = O(\log n)$.

<u>תוחלת זמן הריצה של פעולת הסרה - $\boldsymbol{O(\log n)}$:</u>

1. חיפוש בעלות $O(\log n)$.
2. קישורים בין האיבר הקודם והעוקב של x בעלות של $O(1)$ בכל רמה בה x נמצא, כלומר, העלות הכוללת תהיה $\mathbb{E}(h(x)) = \frac{1}{p}$.

<u>זמן ריצה של פעולות עוקב וקודם - $\boldsymbol{O(1)}$ (*לא* בתוחלת):</u>

בעלות גישה למצביע.