שאלה 1 (15 נקודות)

הוכח \ הפרך: קיים מימוש לערימת מינימום המממש את פעולות הערימה בדרישות הסיבוכיות הבאות:

MakeHeap – יצירת המבנה. סיבוכיות זמן: $O(n)$ במקרה הגרוע.

Insert(x) – הכנסת איבר *x* למבנה. סיבוכיות זמן: $O(\log n)$ במקרה הגרוע.

FindMin – החזרת האיבר המינימלי במבנה. סיבוכיות זמן: $O(1)$ במקרה הגרוע.

DelMin – הוצאת האיבר המינימלי מהמבנה. סיבוכיות זמן: $O(\log n)$ במקרה הגרוע ו-$O(\log\log n)$ משוערך.

<u>שאלה 1</u>

הטענה לא נכונה – **הפרכה**:

נניח בשלילה כי קיים מבנה נתונים כמתואר בשאלה, ונשתמש בו על מנת למיין מערך (מיון בעזרת השוואות) באופן הבא:

1) נכניס את כל האיברים למבנה הנתונים (make heap) בסיבוכיות $O(n)$
2) נפלוט את כל האיברים באופן מסודר בעזרת 2 הפעולות Find Min ו Del Min כל שלב כזה אורך $O(\log(logn))$ משוערך
   *הערה – del min מבוצעת ע"י השוואות ולכן זה מיון בעזרת השוואות
3) נחזור על שלב 2) n פעמים עד לפליטת כל האיברים ממויינים לתוך מערך. $O(n\cdot\log(logn))$ משוערך.

כתוצאה מפעולות אלה מיינו מערך באורך n בסיבוכיות משוערכת של $O(n\cdot\log(logn))$ <u>**בסתירה למה שנלמד בהרצאה**</u>, שקיים חסם תחתון לסיבוכיות הזמן של אלגוריתם מיון מערך בעזרת השוואות, חסם זה הוא $\Omega(n\cdot\log(n))$ במקרה הגרוע ומשוערך.

שאלה 2 (25 נקודות)

בשאלה זו נרצה לתכנן מבני נתונים אשר מאפשרים שמירה של הגרסאות השונות של המבנה לאורך ההיסטוריה. לצורך הפשטות כל הוספת או הסרת איבר מגדירה נקודת זמן בהיסטוריה, כלומר נקודת הזמן *t* היא מיד לאחר ביצוע *t* פעולות הכנסה והוצאה מהמבנה.

<u>שאלה 2</u>

נממש את מבנה הנונים בעזרת Hash Table ממומש ע"י מערך דינמי ופתרון התנגשויות של Chaining Hashing. כל תא ב Hash Table יכיל איבר שהיה בנקודה כלשהי במבנה הנתונים ועבור כל איבר יוצמד לו עץ AVL. העץ יכלול את כל "התקופות" בהם האיבר היה חלק מהמבנה נתונים, כלומר, אינטרוול זמן עבור t שעבור זמנים אלה האיבר היה קיים במבנה הנתונים. המפתח של כל צומת בעץ יהיה הרגע שבו האיבר הוכנס (הקצה השמאלי של אינטרוול הזמן), והצומת גם יכיל את הרגע שבו האיבר הוצא ממבנה הנתונים אם קיים כזה (הקצה הימני של האינטרוול). לדוגמה נניח שאיבר מספר 15 הוכנס למבנה הנתונים בזמן *t=1* והוצא ממנו בזמן *t=3* אז עץ הAVL של האיבר 15 יכיל צומת שהמפתח שלה הוא *t=1* וערך נוסף בצומת יהיה זמן ההוצאה *t=3* כלומר נקבל את האינטרוול [1,3].

עבור רגע שבו איבר מסויים נמצא במבנה הנתונים נרצה לייצג "אינטרוול זמן פתוח" ולכן נגדיר את הקצה הימני כ -1 .

*נשים לב כי לא יכולה להיות חפיפה בין האינטרוולים ולכן ניתן לשמור על העץ תקין.

*סיבוכיות הזמן של פעולות Hash Table הוכחה בהרצאות ובתרגולים ולכן נניח כי ניתן להשתמש ללא הוכחה שלהן.

*למדנו והוכחנו בהרצאות שניתן לאתחל מערך חדש $O(1)$

Init()- נאתחל מערך ריק לHash Table ב$O(1)$.

Insert(x) -

- ניגש לתא של האיבר שאליו מתערבל *X* ב *O(1)* משוערך.
- במידה והאיבר כבר קיים ומצאנו אותו, אנו יודעים כי האיבר קיים במבנה הנתונים או שהיה קיים בעבר.
  - *i.* במידה והאיבר כבר קיים במבנה אז יהיה בעץ צומת עם אינטרוול זמן פתוח (הוא יופיע בצומת הכי ימני תחתון של העץ) נחפש אותו ואם מצאנו ווידאנו שאכן האיבר קיים בנקודת זמן הנוכחית וסיימנו
  - *ii.* אם האיבר היה קיים והוצא בעבר אז לא יהיה בעץ שלו צומת עם אינטרוול זמן פתוח ויש להוסיף צומת עם אינטרוול חדש שמתחיל החל מה- *t* הנוכחי.

  פעולות אלה מתבצעות ב $O(logn_x)$.
- במידה והאיבר לא קיים נכניס אותו לטבלה ב$O(1)$ משוערך. ויש ליצור לו עץ ולהוסיף לעץ שלו צומת עם אינטרוול חדש שמתחיל החל מה- *t* הנוכחי. פעולות אלה מתבצעות ב $O(logn_x)$

Remove(x) - ניגש לאיבר *x* בטבלה ב$O(1)$ משוערך.

- אם האיבר לא קיים סיימנו
- אם האיבר קיים נחפש בעץ שלו ב$O(logn_x)$ אם קיים אינטרוול לא סגור ואם קיים נסגור אותו (כלומר נעדכן את ערך סוף האינטרוול להיות ה*t* הנוכחי). במידה ולא קיים אינטרוול פתוח נסיק כי *x* לא קיים במבנה ב*t* הנוכחי וסיימנו.

Find(x,t) – נחפש את האיבר *x* בטבלה ב$O(1)$ משוערך.

1. במידה והוא לא קיים נחזיר כי לא קיים.
2. במידה וקיים ניגש לעץ שלו ונבצע פעולת חיפוש בעץ למצוא האם הזמן המבוקש *t* נמצא באחד מהאינטרוולים בעץ. החיפוש יבוצע באופן דומה לחיפוש *AVL* רגיל עם ההבדל שההשוואה מתבצעת עם שתי קצוות האינטרוול. עבור כל צומת בחיפוש:
   - *a.* במידה כי *t* נמצע בין תחומי האינטרוול הנוכחי נחזיר כי היה קיים.
   - *b.* במידה כי *t* גדול מהקצה הימני של האינטרוול נמשיך בחיפוש ימינה
   - *c.* במידה כי *t* קטן מהקצה השמאלי של האינטרוול נמשיך בחיפוש שמאלה
   - *d.* אם נגיע לצומת שלא קיים נחזיר כי *x* לא היה קיים בנקודת זמן *t*.

כיוון שכל פעולת ההשוואה נעשית בזמן קבוע $O(1)$, סיבוכיות הזמן של פעולת החיפוש זהה לסיבוכיות בעץ *AVL* רגיל ($O(logn_x)$).

סיבוכיות מקום –

נשים לב כי:

(סך כל פעולות *insert*) ≥ (סך כל הפעולות)   , n מספר האיברים במבנה

לכן $k \geq n$, כאשר k הוא מספר פעולות ההכנסה וההוצאה שבוצעו על כל האיברים במבנה.

נממש את טבלת הערבול באופן דינמי כך שבכל רגע נתון גודל הטבלה הוא $O(n)$

עבור כל איבר קיים עץ שסך כל האיברים בו קטן שווה ל n_x . לכן סכום כל הצמתים בכל העצים של כל האיברים בטבלה הוא לכל היותר k ומכאן שסיבוכיות המקום במקרה הגרוע היא $O(k)$

שאלה 3 (25 נקודות):

בחברת מבני מבוכים© עוסקים בתכנון מבוכים ללקוחות. בחברה זו בונים מבוכים על ידי יצירת מטריצה של $n \times n$ תאים, ואז מסירים באופן הדרגתי קירות בין תאים במבוך. אחד המוצרים אותם מציעה החברה הוא "מבוך חסר מעגלים", במבוך מסוג זה בין כל שני תאים עובר רק מסלול אחד. מוצר נוסף אותו מציעה החברה הוא "מבוך קשיר", במבוך קשיר קיים מסלול בין כל זוג תאים.

הציעו מימוש למבנה נתונים התומך בפעולות הבאות:

$\boldsymbol{Init(n)}$	אתחל מבוך בגודל $n \times n$, בעת האתחול כל הקירות בין שני תאים סמוכים קיימים. סיבוכיות זמן: $O(1)$.
$\boldsymbol{RemoveBarrier((x_1, y_1), (x_2, y_2))}$	הסר את הקיר בין התאים בעלי הקואורדינטות $(x_1, y_1), (x_2, y_2)$. ניתן להניח כי הקלט חוקי ושני התאים הם תאים סמוכים. סיבוכיות זמן: $O(1)$.
$\boldsymbol{IsAcyclic}()$	הפעולה מחזירה האם המבוך הוא מבוך חסר מעגלים. סיבוכיות זמן: $O(n^2 log^*(n^2))$ **במקרה הגרוע**.
$\boldsymbol{IsConnected}()$	הפעולה מחזירה האם המבוך הוא מבוך קשיר. סיבוכיות זמן: $O(n^2 log^*(n^2))$ **במקרה הגרוע**.

שאלה 3

נממש את מבני הנתונים בעזרת *Union-Find* בתוספת השיפורים שנלמדו בתרגולים (איחוד לפי גודל וכיווץ מסלולים) עבור מבנה זה שנלמד והוכח בתרגולים אנחנו יודעים כי סיבוכיות הזמן המשוערכת של פעולת יצירה, חיפוש ואיחוד אורכות $O(\log^* n)$. במקרה של mפעולו

מימוש מבנה הנתונים:

Init() - מאתחלים מטריצה nxn בסיבוכיות $O(1)$ כפי שלמדנו בהרצאה. כל תא במטריצה מכיל 2 ערכים בוליאנים המסמלים 2 קירות של כל תא, אחד יסמל קיר ימני והשני קיר תחתון. המטריצה תהיה מאותחלת ל*false*

False - הקירות קיימים

True – הקירות לא קיימים

RemvoreBarrier – ניגש לאיבר הראשון ב$O(1)$.

1. אם האיבר השני הוא מימין נסיר את הקיר הימני
2. אם האיבר השני הוא מלמטה נסיר את הקיר התחתון
3. במידה והאיבר השני הוא משמאל או מלמעלה נתייחס לאיבר השני כראשון ולראשון כשני ונפעל בהתאם ל 1. ו 2.

IsAcyclic – נאתחל *Union Find* בעל n^2 איברים שכל איבר הוא תא מהמטריצה והתאים הם קבוצות זרות. נעבור איבר איבר במטריצה, על כל קיר לא קיים של 2 איברים סמוכים נאחד אותם עם *union* . רגע לפני שנבצע בין התאים *union* נפעיל עליהם *find* ונשווה את התוצאה. אם נמצא כי התאים הנ"ל נמצאים כבר באותה קבוצה זה אומר שאין ביניהם קיר, וכעת אנחנו מנסים לעשות *union* נוסף ביניהם – כלומר זאת הפעם השניה בסריקה שאנו מנסים לאחד אותם. למעשה נוצר מעגל ונחזיר שהמבוך הוא מעגלי.

במידה והגענו לסוף הטבלה בלי להיתקל ב2 איברים שנמצאים כבר באותה קבוצה, נסיק כי המבוך הוא חסר מעגלים ונחזיר זאת.

אנחנו מתחילים ממבנה נתונים ריק. עבור כל $m = n^2$ פעולות *union*, *find* נשלם $O(m \cdot \log^* n^2)$ ואנחנו מבצעים $2n^2$ פעולות כאלה לכל היותר ולכן הסיבוכיות מקרה גרוע של *IsAcyclic* היא $O(n^2 \cdot \log^* n^2)$ לפי משפט מההרצאה

IsConnected – באופן דומה ל*IsAcyclic* נאתחל *Union Find* , נעבור על כל n^2 האיברים ונשווה כל 2 תאים סמוכים. במידה ו2 תאים אלו נמצאים כבר באותה קבוצה (*find*) נסיק כי הם קשירים. במידה ועברנו על כל התאים ומצאנו 2 תאים במבוך שלא נמצאים באותה קבוצה נסיק כי הם לא קשירים והמבוך עצמו לא קשיר.

אנחנו מתחילים ממבנה נתונים ריק. עבור כל $m = n^2$ פעולות *union*, *find* נשלם $O(m \cdot \log^* n^2)$ ואנחנו מבצעים $2n^2$ פעולות כאלה לכל היותר ולכן הסיבוכיות מקרה גרוע של *IsConnected* היא $O(n^2 \cdot \log^* n^2)$ לפי משפט מההרצאה

שאלה 4 (35 נקודות):

במשחק שבץ-נא (סקראבל) שחקנים נדרשים לשבץ אותיות על לוח משחק כך שיצרו מילים חוקיות. במהלך המשחק שחקן יכול להרחיב מילה ששחקן אחר שם על מנת לקבל מילה אחרת, לכן, כאשר שחקן משבץ מילה הוא לא מעוניין שיהיה אפשר להרחיב אותה למילה אחרת כדי למנוע מהשחקן השני לקבל ניקוד בקלות.

השחקנים במשחק מעוניינים ליצור כלי אשר יזהה מילים שהן תחיליות של מילים אחרות.

שאלה 4

סעיף א.

נממש את הכלי הזה בעזרת מבנה המכיל עץ trie שבכל צומת שלו מכיל את מספר הדולרים בתת העץ שלו (כלומר מספר המחרוזות שהמסלול מהשורש אל הצומת הוא התחילית שלהם). העץ יכלול את כל המחרוזות שהוספו למבנה. בנוסף המבנה נתונים ישמור את מספר זוגות המחרוזות במבנה שאחד מהם תחילית של השני (להלן יקרה "מספר הזוגות").

Init() - נאתחל עץ trie ריק בO(1) ואת מספר הזוגות ב0.

Insert(s) - נכניס את המחרוזת s לעץ בדומה לאיך שנלמד בהרצאה בO(|s|) עם כמה תוספות

1. לאורך המסלול נעדכן בכל צומת את מספר המחרוזות השמור בצומת להיות 1 יותר. כיוון שהמסלול הוא באורך |s| וכל פעולה נעשית בזמן קבוע זה יעשה בO(|s|).

2. לאורך המסלול עבור כל צומת שנעבור שתצא ממנו קשת ל$ נעלה את "מספר הזוגות" ב1. כל מקרה כזה למעשה מייצג מחרוזת קיימת בעץ שהיא תחילית של המחרוזת שאנחנו מכניסים לתוך העץ. כיוון שהמסלול הוא באורך |s| וכל פעולה נעשית בזמן קבוע זה יעשה בO(|s|).

3. נוסיף אל "מספר הזוגות" את מספר המחרוזות שהמחרוזת שנכניס לעץ היא תת מחרוזת שלהם. מספר זה למעשה שמור בצומת שמוביל לצומת ה$ של המחרוזת שאנחנו מכניסים לעץ. הפעולה נעשית בזמן קבוע כלומר בO(1).

סך הכל הסיבוכיות הכוללת O(|s|) כנדרש.

Remove(s) - נבצע הסרת מחרוזת בO(|s|) בדומה לאיך שנלמד עם כמה תופסות

1. לאורך המסלול נעדכן בכל צומת את מספר המחרוזות השמור בצומת להיות 1 פחות. כיוון ש המסלול הוא באורך |s| וכל פעולה נעשית בזמן קבוע זה יעשה בO(|s|).

2. לאורך המסלול עבור כל צומת שנעבור שתצא ממנו קשת ל$ נחסר מ "מספר הזוגות" אחד. כל מקרה כזה למעשה מייצג מחרוזת קיימת בעץ שהיא תחילית של המחרוזת שאנחנו מוציאים מהעץ. כיוון שהמסלול הוא באורך |s| וכל פעולה נעשית בזמן קבוע זה יעשה בO(|s|).

3. נחסר מ "מספר הזוגות" את מספר המחרוזות שהמחרוזת שנכניס לעץ היא תת מחרוזת שלהם. מספר זה למעשה שמור בצומת שמוביל לצומת ה$ של המחרוזת שאנחנו מוציאים מהעץ. הפעולה נעשית בזמן קבוע כלומר ב$O(1)$

PrefixExists() - אם מספר הזוגות הוא לא 0, נחזיר שקיים זוג מחרוזות כזה. $O(1)$

<u>סעיף ב.</u>

במקום לשמור את "מספר הזוגות" מחוץ לעץ נשמור אותו בתוך כל צומת בעץ. מספר זה ייצג את כל הזוגות מחרוזות שהם מסלול מהשורש אל סימן $ בתת עץ של הצומת שהם תת מחרוזת אחד של השני.

נשנה את insert כך:

2. במקום סעיף 2 המקורי נעבור על המסלול שהוספנו בסדר הפוך (כלומר מהדולר ועד לשורש). נשמור משתנה זמני שסופר את כמות הפעמים שנתקענו בקשת $ שיוצאת מהצומת שאנחנו נמצאים בו. בכל צומת שאנחנו עוברים נוסיף את המשתנה הזמני

ל"מספר הזוגות". כך למעשה נעדכן עבור כל צומת במסלול את מספר הזוגות בתת עץ שלו באופן תקני ב-O(|s|).

3. נוסיף את המספר מסעיף ה3 המקורי לכל אורך המסלול מהשורש ואל סוף המילה שהכנסנו. גם עדכון זה ב-O(|s|).

סך הכל הסיבוכיות לא השתנתה.

נשנה את remove כך:

2. במקום 2 המקורי נעבור על המסלול שהוספנו בסדר הפוך. נשמור משתנה זמני שסופר את כמות הפעמים שנתקענו בקשת $ שיוצאת מהצומת שאנחנו נמצאים בו. בכל צומת שאנחנו עוברים נחסר מ"מספר הזוגות את המשתנה הזמני. כך למעשה נעדכן עבור כל צומת במסלול את מספר הזוגות בתת עץ שלו באופן תקני ב-O(|s|).

3. נחסר את המספר מסעיף 3 המקורי לכל אורך המסלול מהשורש ועד סוף המילה שהכנסנו.

סך הכל הסיבוכיות לא השתנתה.

PrefixExists(s)

נסייר בעץ עד לצומת שיוצאת ממנו קשת ה$ של המילה s ב-O(|s|).

- אם לא קיימת צומת כזאת אז נחזיר שלא קיים זוג כזה
- אם מספר הזוגות בצומת שהגענו אליו אינו 0 נחזיר שקיים זוג מחרוזות כזה.
- אם הוא 0 נחזיר שלא קיים