נושאים מתקדמים במבני נתונים – תרגיל 4

תרגיל 1

<u>צ"ל</u>: להוסיף יכולת לאלגוריתם של Thorup and Zwick להחזיר את המסלול בין שני הקודקודים בנוסף למרחק.

<u>פתרון</u>:

ה-preprocessing הורכב משני חלקים:

1. הרצת דייקסטרה מ-super source כדי למצוא את כל ה-pivot-ים. ביצענו k הרצות (אחד עבור כל רמה i כדי למצוא את ה-pivot-ים מרמה i – לכל צומת v את $p_i(v)$). בזמן הריצה כאשר הגענו לצומת v קבענו ש-pivot שלו הוא ה-p_i שעברנו דרכו ישר אחרי ה-super source. בדרך למעשה הרכבנו עץ של המסלולים הקצרים ביותר מכל p_i לכל ה-v שהוא ה-pivot שלהם מרמה i.
   השינוי שנכניס יהיה לשמור עבור כל אחת מההרצות את העץ הנ"ל. גודל העץ של הרצה בודדת הוא $O(n)$ וסה"כ עבור k ההרצות נזדקק ל- $O(kn)$ מקום.
2. הרצת דייקסטרה (מיוחדת) לכל $v \in V$ כדי למצוא את $B(v)$. עשינו זאת בצורה עקיפה ע"י חישוב cluster-ים. כלומר כל צומת "אסף" את כל הצמתים שהוא שייך ל-bunch שלהם.
   לפי הגדרה לכל $w \in A_i - A_{i+1}$: $C(w) = \{v \in V | \delta(w,v) < \delta(v, p_{i+1}(v))\}$.
   הייחוד בדייקסטרה שהורצה היה שקודקוד הוכנס לערמה רק אם אורך המסלול מ-w אליו היה קצר מאשר ל-pivot כלשהו מרמה $i+1$, כלומר אספנו בדיוק את כל ה-v לפי ההגדרה שלעיל.
   תוך כדי הריצה למעשה בנינו עץ מסלולים הכי קצרים מכל קודקוד w לכל ה-v שב-cluster שלו. השינוי שנעשה באלגוריתם יהיה לשמור את העץ הזה. גודל כל עץ פרופורציאונלי לגודל ה-cluster וגודל כל ה-cluster-ים הוא פרופורציאונלי לגודל כל ה-bunch-ים שהוא $O(kn^{\frac{1}{k}})$.

סה"כ המקום הנוסף שאנו זקוקים לו הוא $O(kn^{\frac{1}{k}})$. מאחר וזוהי דרישת המקום מהאלגוריתם המקורי, לא שינינו את דרישות המקום.

ה-query עובד בצורה שמחזירים את סכום המרחקים בין צומת אחת ל-pivot שלו (שמרנו את המרחק והמסלול בשלב ה-preprocessing הראשון) ובין ה-pivot לצומת השני כאשר ה-pivot נמצא ב-bunch של הצומת השני (את המרחק והמסלול הזה שמרנו בשלב ה-preprocessing השני).
על מנת להחזיר את המסלול נשתמש בשני חלקי המסלול ששמרנו בשלב ה-preprocessing.
סיבוכיות זמן הריצה היא כסיבוכיות זמן הריצה הקודמת ועוד $O(1)$ לכל קשת במסלול.

תרגיל 2

צ"ל: $\delta(u,p_i(v)) + \delta(v,p_i(v)) \le (4i+1)\delta(u,v)$ כאשר i האינדקס המינימלי כך ש-$p_i(v) \in B(u)$

הוכחה:

נתבונן ב-pivot כלשהו של $p_j - v$ (לכל $j \in [1,k-1]$) ובמשולש הנוצר בין $u,v,p_j(v)$.

מאי-שיוויון המשולש נובע ש- $\delta\left(u,p_j(v)\right) \le \delta\left(v,p_j(v)\right) + \delta(u,v)$ ואם נציב זאת בטענה שעלינו הוכיח נקבל ש-$\delta(u,p_i(v)) + \delta(v,p_i(v)) \le 2\delta\left(v,p_j(v)\right) + \delta(u,v)$.

נרצה כעת להוכיח טענת עזר.

טענה: לכל $j \in [1,k-1]$ אם $p_{j-1}(v) \notin B(u)$ אזי $\delta\left(v,p_j(v)\right) \le \delta\left(v,p_{j-1}(v)\right) + 2\delta(u,v)$

לשם השלמות נגדיר ש- $p_0(v) = v$

הוכחה:

יהי j כנ"ל, כלומר $p_{j-1}(v) \notin B(u)$. כמו כן נגדיר כ-$D(u)$ (diameter) את המרחק הגדול ביותר בין u לצומת אחר ב-bunch שלו.

מכאן נובע ש-

$$\boldsymbol{\delta\left(v,p_j(v)\right)} \le^1 \delta\left(v,p_j(u)\right) \le^2 \delta(u,v) + \delta\left(u,p_j(u)\right) \le^3 \delta(u,v) + D(u) \le^4 \delta(u,v) + \delta\left(u,p_{j-1}(v)\right) \le^5 \boldsymbol{\delta\left(v,p_{j-1}(v)\right) + 2\delta(u,v)}$$

1: לפי הגדרת p_j מתוך A_j הוא הצומת עם המרחק המינימלי עד ל-v

2: אי שיוויון המשולש במשולש $v,u,p_j(u)$

3: לפי הגדרת $D(u)$ והעובדה ש-$p_j(u) \in B(u)$ (לפי הגדרת $p_j(u)$)

4: מאחר ו-$p_{j-1}(v) \notin B(u)$ לפי הגדרת $D(u)$ מתקיים $D(u) \le \delta\left(v,p_{-1j}(v)\right)$

5: אי שיוויון המשולש במשולש $u,v,p_{j-1}(v)$

כעת נשתמש בטענת העזר על הביטוי $\delta\left(v,p_j(v)\right)$ על מנת להוכיח את הטענה הנדרשת. ניתן לעשות זאת מאחר ולכל $j \in [1,i]$ מתקיים $p_{j-1}(v) \notin B(u)$.

$$\delta(u,p_i(v)) + \delta(v,p_i(v)) \le 2\delta(v,p_i(v)) + \delta(u,v) \le 2(\delta(v,p_{i-1}(v)) + 2\delta(u,v)) + \delta(u,v)$$
$$\le \cdots \le 2\left(\delta(v,p_0(v)) + 2i \cdot \delta(u,v)\right) + \delta(u,v) = (4i+1)\delta(u,v)$$

תרגיל 3

<u>צ"ל</u>: אלגוריתם למציאה לכל מקום i במחרוזת T באורך n, אינדקס k ואורך ℓ בתורך מחרוזת P כך שיש והתאמה באורך ℓ במקומות i ו-k ב-T ו-P בהתאמה.

<u>פתרון</u>:

האלגוריתם:

1. נבנה Generalized Suffix Tree עבור שתי המחרוזות (כמו שעשינו בכיתה בבעיית ה-LCS)
   זמן ריצה: $O(n+m)$ ע"י האלגוריתם של Ukkonen.
2. עבור כל סיפא של P_k (שמיוצגת ע"י עלה בעץ) נעלה במעלה העץ עד לשורש ועבור כל צומת פנימי נרשום את מספר הסיפא k (אינדקס ההתחלה שלה ב-P) ואת כמות התווים שנשארו עד השורש - ℓ (אנו יודעים זאת ממספר הסיפא, האורך שלה וכמות התווים עליהם דילגנו בהליכה במעלה העץ). לאחר ביקור בצומת פנימי נסמן אותו כדי לדעת לעצור בו אם נגיע אליו מסיפא אחרת.
   זמן ריצה: נעבור על כל קשת בעץ לכל היותר פעם אחת ויש סה"כ $O(n+m)$ קשתות כנ"ל. כלומר זמן הריצה הוא $O(n+m)$.
3. עבור כל סיפא של T_i (שמיוצגת ע"י עלה בעץ) נעלה במעלה העץ עד אשר נתקל בצומת פנימי בו שמור מידע על P_k כלשהו באורך ℓ**.** תמיד קיים צומת כזה כי השורש מייצג תת מחרוזת של P באורך 0. כאשר נגיע לצומת הנ"ל נדע שלאינדקס i ב- T מתאימה תת מחרוזת באורך ℓ החל מאינדקס k ב-P. כדי למנוע מעבר חוזר על קשתות בעץ עבור סיפות אחרות של T נרד חזרה עד לעלה ממנו הגענו ונעתיק את המידע k, ℓ לכל הצמתים הפנימיים בדרך.
   זמן ריצה: נעבור על כל קשת בעץ לכל היותר פעמיים, פעם אחת בעלייה (לכל היותר עד לשורש) ופעם אחת בירידה עד לעלה בתהליך ההעתקה. מאחר ויש סה"כ $O(n+m)$ קשתות בעץ זמן הריצה הוא $O(n+m)$.

נכונות:

בשלב 2 אנו מגלים את ה-Longest Common Prefix (LCP) של סיפא של P ושל סיפא של T. זו לפי הגדרה תת-המחרוזת הארוכה ביותר המשותפת ל-P ו-T החל מנקודות ההתחלה של הסיפות של P ו-T בהתאמה. אם הצומת הפנימי בו אנו שומרים את k, ℓ הוא נקודת פיצול בין סיפות של P ו-T אז מהגדרת ה-Suffix Tree הוא מייצג תחילית משותפת ל-P ו-T (המחרוזת מהשורש עד לאותו הצומת). אם הצומת הוא לא נקודת פיצול אז המידע יהיה רלוונטי לצומת שמעליו עם קיזוז אורך המחרוזת על הקשתות שעוברים בדרך.
מאחר ואנו עוצרים אם כבר סימנו צומת אז בהכרח הנקודה הנמוכה ביותר בעץ היא ה-LCP.
בשלב 3 ההעתקה של המידע משמרת את הנכונות ובאה לשמור על חסם זמן הריצה.

זמן ריצה:

סה"כ זמן הריצה הוא $O(n+m)$.

תרגיל 4

סעיף 1

<u>צ"ל</u>: אלגוריתם לבניית עץ קרטזי.

<u>פתרון</u>:

נגדיר אלגוריתם שיבצע מעבר בודד על רצף המספרים $i_1, i_2, \dots, i_n$ כך כאשר האלגוריתם יסיים לעבור על המספר ה-j יהיה בנוי העץ קרטזי $i_1, i_2, \dots, i_j$.

האלגוריתם:

נתחיל מעץ המכיל את האיבר הראשון (i_1).

יהי i_j האיבר האחרון שהכנסנו לעץ (תמיד נזכור מצביע אליו). כעת נרצה להכניס את האיבר ה-i_{j+1}.

1. אם $i_{j+1} \geq i_j$ אזי נוסיף אותו לעץ כבן הימני של i_j.
2. אחרת, יהי $P(i_j)$ האבא של i_j בעץ ויהי u הבן הימני של $P(i_j)$ אם $P(i_j) \leq i_{j+1}$ אז נהפוך את i_{j+1} להיות הבן הימני של $P(i_j)$ ואת u להיות הבן השמאלי של i_{j+1}. אם $P(i_j) > i_{j+1}$ אז נמשיך ל-$P\left(P(i_j)\right)$ וכן הלאה.
3. אם הגענו לשורש העץ ועדיין הוא גדול מ-i_{j+1} אז נהפוך את i_{j+1} להיות השורש החדש ואת השורש הישן לבן השמאלי שלו.

נכונות:

נוכיח את נכונות האלגוריתם באינדוקציה על j (כמות המספרים ברצף שהאלגוריתם כבר בנה עבורם עץ קרטזי).

עבור $j = 0$ העץ הריק הוא העץ הנכון ועבור $j = 1$ העץ הנכון הוא האיבר הבודד.

נניח את נכונות האלגוריתם עבור j ונוכיח שלאחר הוספת האיבר i_{j+1} העץ שנוצר הוא עץ קרטזי תקין עבור $j + 1$ מספרים.

נשים לב:

- לאיבר ה-j אין בן ימני. זאת מאחר והוא האיבר האחרון שהוכנס ולפי האלגוריתם באף אחד מהמקרים לא יכול להיות לו בן ימני.
- האיבר i_j הוא הבן הימני של האבא שלו (אם יש לו אבא). במקרה הראשון בקוד הוא מתווסף כבן הימני. במקרה השני הוא גם כן תופס את המקום של הבן הימני. אותו טיעון תופס רקורסיבית עד לשורש. לא ייתכן ש- i_j הוא בן שמאלי כלשהו כי זה אומר שהוא נמצא ברצף המספרים משמאל לאיבר אחר אבל לפי הגדרה הוא האחרון (עד כה).

נעבור על המקרים לפי החלוקה שבאלגוריתם:

1. $i_{j+1} \geq i_j$ - לפי הגדרה i_{j+1} אמור להיות בתת העץ הימני של האיבר המינימלי ברצף עד i_j. מאחר והוספנו את i_{j+1} כבן ימני של האיבר האחרון שהוספנו ואנו יודעים שעד לשלב זה העץ הוא עץ קרטזי תקני (כלומר i_j נמצא בתת העץ הימני של האיבר המינימלי ברצף עד i_{j-1}) אזי התנאי מתקיים גם עבור i_{j+1}.
2. $i_{j+1} < i_j$ ומצאנו צומת w שהוא אב קדמון של i_j כך ש $i_{j+1} \geq w$ - במקרה זה האלגוריתם עושה שני צעדים:
   a. לוקח את תת העץ הימני של w והופך אותו לבן השמאלי של i_{j+1}. צעד זה שומר על התכונה ש-i_{j+1} קטן מכל תת העץ השמאלי שלו וכל האיברים ששם באו לפניו ברצף. כמו כן מאחר ולקחנו את תת העץ כמו שהוא לא פגענו בתכונות שלו כעץ קרטזי.
   b. הופך את i_{j+1} לבן הימני של w. צעד זה שומר על התכונה ש- i_{j+1} גדול מהאבא שלו ומאחר והוא בא אחריו ברץ המספרים הוא אכן נמצא מימינו. כמו כן לפי הדרך שבה טיילנו בעץ עד שמצאנו את w אנו יודעים ש-w הוא האיבר הראשון שקטן מ-i_{j+1} ולכן i_{j+1} הוא המינימום בעץ שהוא מהווה השורש שלו.

מכאן שצעד זה שומר על כל האינוריאנטות של עץ קרטזי.

3. $i_{j+1} < i_j$ ולא מצאנו צומת w שהוא אב קדמון של i_j כך ש $i_{j+1} \geq w$ - במקרה זה האלגוריתם מייצר שורש חדש ושם בו את i_{j+1}. צעד זה שומר על תכונות העץ הקרטזי כי i_{j+1} הוא האיבר הקטן ביותר ברצף עד כה (מהעובדה שהגענו עד לשורש). כמו כן כל האיברים שהיו ברצף עד כה נמצאים בתת העץ השמאלי. מכאן שקיבלנו עץ קרטזי כולל i_{j+1}.

זמן ריצה:

נראה שהאלגוריתם רץ בזמן $O(n)$. נתבונן בעומק של האיבר האחרון שהכנסנו (כאשר השורש בעומק 0 ומשם העומק רק גדל) ונסמן אותו ב-d. כמו כן כאשר אנו עולים בעץ במקרה 2 נסתכל על זה כאילו האב הקדמון שאנו בוחנים הוא האיבר האחרון ו-d יהיו עומקו.

ב-1.

נפרק את הניתוח לפי שלושת המקרים באלגוריתם:

- במקרים 1 ו-2 העומק d גדל ב-1 כי אנו מוסיפים בן לאיבר שהתבוננו בו מקודם.
- במקרה 3 העומק d נשאר 0 כי היינו בשורש ונשארנו בשורש.

מאחר ויש סה"כ n איברים העומק d יכול לגדול לכל היותר n פעמים. מכאן שבטיפוסים במעלה העץ במקרה 2 אנו יוכלים להקטין את d לכל היותר n פעמים. מעבר לכך בכל איטרציה מתבצעת כמות קבועה של פעולות. על כן האלגוריתם רץ בזמן $O(n)$.

סעיף 2

<u>צ"ל</u>: מבנה נתונים לבעיית ה-Range Minimal Query המשתמש בעץ קרטזי.

<u>פתרון</u>:

בבעיית ה-RMQ נרצה בהנתן מערך A ותחום $[i,j]$ להחזיר את האיבר המינימלי ב-$A[i:j]$ ואת האינדקס שלו ב-A. נעשה זאת ע"י כך שנבנה עץ קרטזי ל-A (מה שיקח $O(n)$ זמן ומקום). נשים לב שה-LCA של $A[i]$ ו-$A[j]$ בעץ הקרטזי (נסמנו $A[k]$) הוא בדיוק האיבר המינימלי בתחום $[i,j]$ (זאת מהגדרת העץ הקרטזי).

על מנת לענות על שאילתת ה-RMQ פשוט נחזיר את ה-LCA של $A[i]$ ו-$A[j]$ בעץ הקרטזי. על מנת שנוכל להחזיר גם את האינדקס של האיבר נצטרך לשמור מידע זה בעץ בניית העץ הקרטזי (ניתן לעשות זאת בלי פגיעה בחסמי זמן הריצה או המקום).

דבר זה מחזיר אותנו לפתרון בעיית ה-LCA שדנו בה בכיתה אותה פתרנו בשימוש ב-$O(n)$ מקום עם זמן הריצה של $O(1)$.

תרגיל 5

סעיף 1

<u>צ"ל</u>: בהנתן פרמוטציה π של המספרים $1,2 \dots, n$ ישנה מחרוזת s באורך n ש-π הוא ה-Suffix Array של s.

<u>הוכחה</u>:

תהי $\pi = i_1, i_2, \dots, i_n$ פרמוטציה של המספרים $1,2 \dots, n$. נבנה את המחרוזת s באמצעות π. הא"ב אשר נשתמש בו יהיה $\Sigma = \{1,2, \dots, n\}$ כאשר $1 < 2 < \cdots < n$. המחרוזת s תוגדר כך שהאות ה-i_j תהיה j.

נשים לב שבצורה זו ה-Suffix Array יתחיל ב-i_1 כי האות במקום זה היא 1 והסיפא הבאה תהיה i_2 כי היא תתחיל ב-2 ולפי הסדר שהגדרנו $1 < 2$. בצורה זו ה-Suffix Array של s הוא בדיוק π. הוכחנו שלכל π שהוא קיימת s באורך n (האורך של s הוא כאורך של π שהוא n) כך ש-π הוא ה-Suffix Array של s.

סעיף 2

1. נבחר א"ב $\Sigma = \{a\}$. המחרוזת המתאימה היא: $\underbrace{aaaaaa}_{n}$ (n פעמים).
   ה-Suffix Array הנתון אכן מתאים לה $(n, n-1, \dots 2,1)$ מאחר ו-$a < aa < aaa < \cdots < \underbrace{aaaaaa}_{n}$.
   לא תיתכן מחרוזת מעל א"ב קטן יותר מאחר והא"ב כבר מינימליסטי (סימן אחד).
2. נבחר א"ב $\Sigma = \{1,2\}$ והמחרוזת המתאימה היא: $1111 \dots 2$.
   ראינו בסעיף הקודם שאם הא"ב מורכב מאות אחת אז לא נוכל להרכיב מחרוזת שתתן לנו את ה-Suffix Array $(1,2, \dots, n-1, n)$. על כן נזדקק לפחות לשתי אותיות, נראה שזה מספיק. הסיפא הראשונה יותר קטנה מהסיפא הבאה כי במקום שבראשונה יש 1 בשניה יש 2. כנ"ל לגבי ההמשך.
3. נבנה את הא"ב והמחרוזת בהתאם ל-Suffix Array. נתחיל מא"ב מינימלי המכיל את האות a. ה-Suffix Array שלנו הוא $n, 1, n-1, 2, n-3, 3, \dots, n/2+1, n/2$. מכך שהסיפא הראשונה במערך היא הסיפא האחרונה נסיק שהאות האחרונה במחרוזת היא a.
   נבחן כעת שתי אופציות:
   - האות הראשונה במחרוזת היא גם a: נוכל לשים את האות הזאת גם גאות הראשונה, זאת מאחר ו-$a < aa \dots$. לפי ה-Suffix Array אנו רואים שהסיפא הבאה לפי הסדר הלקסיקורגפי היא הסיפא ה-$n-1$. לא ייתכן שאות זו היא a כי אז הסיפא ה-$n-1$ תהיה קטנה מהסיפא ה-1. זאת כי הסיפא ה-$n-1$ היא רק aa בעוד שהסיפא ה-1 רק מתחילה ב-aa ומשיכה בעוד אותיות. האות ה-$n-1$ לא יכולה להיות קטנה מ-a (נתייחס כרגע ל-a כאות הכי גדולה בא"ב שהשתמשנו בה לאחרונה בבניית המילה) כי אז סיפא זאת שוב תהיה קטנה מהסיפא ה-1. מכאן שעלינו לבחור אות חדשה גדולה מ-a להוסיפה לא"ב.
     באופן רקורסיבי נוכל להשתמש באות הזאת כאות ה-2 להמשיך באותם הטיעונים לגבי האותיות ה-$n-3$ וה-3 עד ל-$n/2+1$ ו-$n/2$. בצורה זו נקבל א"ב בגודל $n/2$ המכיל את בה"כ את האותיות $n/2 \dots 1$ (נעבור לסימון מספרי). ואת המחרוזת שהיא הפלינדרום שמכיל כ-$n/2$ האותיות הראשונות את הא"ב לפי סדר עולה.
   - האות הראשונה היא לא a: לא נוכל לבחור אות שקטנה מ-a כי אז הסיפא ה-1 תהיה קטנה מהסיפא ה-n. נותר לנו לבחור אות גדולה מ-a, למשל b. נמשיך לאות ה-$n-1$. היא לא יכולה להיות a (או כל אות שקטנה מ-a) כי אז הסיפא ה-$n-1$ תהיה קטנה מהסיפא ה-1 ($a < b$). כלומר היא תוכל להיות או b או אות חדשה הגדולה מ-b. בכל מקרה באסטרטגיה זו לא הרווחנו א"ב קטן יותר מאשר באופציה הקודמת על כן נדבוק בה.