נושאים מתקדמים במבנ״ת ואלגוריתמים
תרגיל 3

שאלה 1

T יער פורש של הגרף. נייצג את T באמצעות E - יער Euler tour forest של T.
לכל קודקוד $v \in V$, נגדיר:

- $C(v)$ רכיב הקשירות בגרף בו v נמצא
- $T(v)$ העץ הפורש (כרגע) של $C(v)$

מבנה הנתונים שלנו ישמור, לכל קודקוד $v \in V$, את $E(v)$: עותק מלא של E, כאשר הערך לכל קודקוד u ב$E(v)$ הוא $w(u,v)$ (או ∞ אם הקשת (u,v) לא קיימת).

בתחילת האלגוריתם אין קשתות. כל רכיבי הקשירות והעצים הפורשים ביער הם סינגלטונים.

<u>הוספת קשתות:</u>
מכניסים קשת (u,v). ראשית, נעדכן את ערך u ב$E(v)$ ואת ערך v ב$E(u)$ להיות $w(u,v)$.
כעת, נפריד למקרים:
$C(u) \neq C(v)$: אז (u,v) קשת עץ שמאחדת את $T(v), T(u)$ לעץ משותף T. לכל $x \in V$ נבצע פעולת $link(u,v)$ ב$E(x)$.
$C(u) = C(v)$: נסמן $p_T(u,v)$ המסלול על קשתות העץ $T(v) = T(u)$ בין u, v, $e_T(u,v)$ הקשת הכבדה ביותר על המסלול. אם $w(u,v) \leq w\left(e_T(u,v)\right)$ אז (u,v) כבדה מכל קשת על המסלול $p_T(u,v)$, ואין צורך לשנות את היער הפורש וסיימנו. אחרת, $w(u,v) < w\left(e_T(u,v)\right)$ ואז (u,v) קלה לפחות מ$e_T(u,v)$ והיא תחליף אותה בתוך קשת עץ ב$T(u) = T(v)$. לכל $x \in V$ נבצע שתי פעולות על $E(x)$: נתחיל בפעולת $cut\left(e_T(u,v)\right)$ שתוציא את הקשת שמחליפים מהיער הפורש, ולאחר מכן פעולת $join(u,v)$ שתכניס את (u,v) כקשת עץ.

<u>הסרת קשתות:</u>
מוציאים קשת (u,v). ראשית, נעדכן את ערך u ב$E(v)$ ואת ערך v ב$E(u)$ להיות ∞.
אם (u,v) לא קשת עץ אז אין הבדל בT ונסיים.
אחרת, u, v באותו רכיב קשירות C_{uv} עם עץ פורש T_{uv}. בכל קודקוד $x \in V$ נבצע פעולת $cut(u,v)$ ב$E(x)$. פעולת cut מפצלת את T_{uv} לשני תתי עצים T_u, T_v. נרצה לבדוק האם יש קשת אחרת שמחברת קודקוד $u' \in T_u$ לקודקוד $v' \in T_v$, ואם כן אז למצוא את הקלה ביותר שעושה זאת ולהוסיף אותה ליער הפורש. אז מחפשים קשת (u', v') עם $u' \in T_u, v' \in T_v$ במשק מינימלי. עוברים על כל הקודקודים $u' \in T_u$. לכל קודקוד $u' \in T_u$ נבצע פעולת $findmin(T_v)$ ב$E(u)$, שתתן את הקודקוד $v' \in T_v$ עם $w(u', v')$ מינימלי (מבין כל השכנים של u' בT_v), או ∞ אם אין כאלה. נקבל סט של זוגות (u', v') כאלה, נמצא את הזוג עם המשקל המינימלי $w(v', u')$. אם המשקל המינימלי הוא ∞ אז אין קשת שמחברת את T_u, T_v ולכן סיימנו. אחרת מצאנו קשת מחליפה (u', v'), ולכל $x \in V$ נבצע פעולת $link(u', v')$ ב$E(x)$.

<u>נכונות:</u> נובעת ישירות מצעדי האלגוריתם. בתחילת הריצה E יער פורש של סינגלטונים. בכל שלב שבו משנים את מבנה היער מעדכנים את $E(v)$ על כל קודקוד $v \in V$, ולכן מבנה $E(v)$ זהה בכל הקודקודים ומתאר את T.
T יער פורש מינימלי: ראינו בשיעור הסבר לכך שהמימוש הנ"ל לפעולות שומר על מינימליות היער.
נחזור עליו בקצרה:

- הוספת קשת (u,v): אם u, v ברכיבי קשירות שונים לפני ההוספה אז (u,v) הקשת היחידה שמחברת בין רכיבי הקשירות, ולכן היא חייבת להפוך להיות קשת עץ. אחרת, (u,v) הופכת

לקשת עץ רק אם קיימת קשת עץ במסלול $u \to v$ בעץ הכבדה מ(u,v), ואז (u,v) מחליפה את אותה הקשת הכבדה ביותר.

- הסרת קשת (u,v): אם היא לא קשת עץ אז ברור שאין השפעה על העץ. אם היא כן, אז ייתכן שיש קשת אחרת שמחליפה אותה. מספיק לחפש בקשתות המחברות בין קודקודים בשני תתי העצים החדשים, ולמצוא את המינימלית מביניהן.

<u>סיבוכיות מקום:</u> מבנה הנתונים מחזיק עותק מלא של Euler tour forest על קשתות T לכל קודקוד $v \in V$. אם $|V| = n$, אז מספר הקשתות בT הוא לכל היותר $n-1$. גודל E לינארי במספר הקשתות, כלומר לינארי בn. בסה"כ שומרים מבנה בגודל $O(n)$ לכל קודקוד מתוך n הקודקודים, והמקום הכולל $O(n^2)$.

<u>סיבוכיות זמן:</u>

בנייה ראשונית: לכל קודקוד $v \in V$ בונים את $E(v)$ שבתור התחלה, כל הערכים בו ∞. הזמן פרופורציונלי לגודל מבנה הנתונים, $O(n^2)$.

הכנסת קשת (u,v):

- עדכון ערך u ב$E(v)$ ולהפך – $O(\log n)$
- בדיקת רכיבי הקשירות: שקולה ל2 פעולות find-tree על E (ניתן לבצע אותה בכל $E(x)$ שנבחר, למשל ב$E(v)$). כל פעולה לוקחת $O(\log n)$.
- אם רכיבי הקשירות שונים, עושים $O(n)$ פעולות $link(u,v)$ (אחת לכל $E(x)$). כל פעולת $link$ בעלות $O(\log n)$.
- אם רכיבי הקשירות זהים, מוצאים את המסלול המשותף בין u,v ואת הקשת המקסימלית עליו בזמן $O(n)$. אם $w(u,v)$ גדול ממשקל הקשת המקסימלית סיימנו. אחרת, מבצעים $O(n)$ פעולות $cut\left(e_T(u,v)\right)$ ועוד $O(n)$ פעולות $link(u,v)$ - כ"א בזמן $O(\log n)$ ובסה"כ $O(n \log n)$.

בסה"כ, אם (u,v) הופכת להיות קשת עץ העלות תהיה $O(n \log n)$. אחרת העלות $O(n)$.

הוצאת קשת (u,v):

- עדכון ערך u ב$E(v)$ ולהפך – $O(\log n)$
- אם (u,v) לא קשת עץ - סיימנו.
- אם (u,v) קשת עץ, מתחילים ב$O(n)$ פעולות cut, כ"א בזמן $O(\log n)$. אחרת כך עוברים על T_u עם $O(n)$ קודקודים, ולכל אחד מבצעים פעולת $findmin(T_v)$ בזמן $O(\log n)$. לבסוף, אם מצאנו קשת מחליפה, מבצעים $O(n)$ פעולות $link$, כ"א בזמן $O(\log n)$.

בסה"כ, אם (u,v) לא קשת עץ העלות $O(\log n)$, ואחרת $O(n \log n)$.

שאלה 2

<u>אינטואיציה:</u> ניזכר באנליזה של אלגוריתם העצים והצפיפות שראינו בשיעור. ראינו שאם מבצעים relabel בצומת בעץ, אז אפשר לחייב את הפעולה על פני על האיברים שנוספו ומילאו עלים מאז הrelabel האחרון באותו צומת, ואז כל צומת משלם בהכנסה על $\log n$ הורים בעץ בהם עלול להתבצע relabel בעתיד. אם נכפיל פי $\log n$ את מספר הצמתים שנכנסים בין זוג פעולות relabel על אותו צומת, אז כל איבר יוכל לשלם רק על אחד מבין $\log n$ ההורים במסלול והעלות הamortized תהיה קבועה.

נסמן בL את רשימת כל n האיברים הממויינים ע"פ הסדר. נאכסן את L במבנה נתונים בשתי רמות באופן הבא:
רמה 1: נחלק את L לλ תתי רשימות $l_1, \dots, l_\lambda$ כך שהאיברים בכל תת רשימה ממוינים ביניהם, וקבוצת האינטרוולים הנקבעים ע"י הרשימות $\{I(l) = [\min(l), \max(l)] : l \in L\}$ מקיימת שכל זוג אינטרוולים $I(l_1), I(l_2)$ עם l_1, l_2 זרים זה לזה ($I(l_1) \cap I(l_2) = \phi$).
בתוך תתי הרשימות, נמיין איברים באופן הבא: כל איבר $x \in L$, יזכור את $l(x)$ = תת הרשימה היחידה שבה x ממוקם. ניתן תגית מספרית $label(x)$ שתגדיר את המיקום של x בתוך $l(x)$.
בהכנסה של איבר חדש y לתוך תת רשימה l, ניתן תגית $l(y)$ המבוססת על ממוצע בין הקודם לy ובין העוקב אחריו בl (או תגית גדולה ב1 מהקודם, אם y המקסימלי בl).
נגדיר k פונקציה כלשהי של n. האלגוריתם ישמור על כך שבמהלך ריצת האלגוריתם, גודל כל אחת מתתי הרשימות l_i הוא "בערך" k, ובפועל – לא יותר מ$2k$. בכל פעם שרשימה כלשהי תגדל מדי, נפצל אותה לתתי רשימות באורך k ונכניס את תתי הרשימות החדשות לעץ.

רמה 2: מגדירים יחס סדר על הרשימות $\{l_i\}$: לכל זוג תתי רשימות l_1, l_2 נגדיר $l_1 < l_2$ אם $\forall x_1 \in l_1, x_2 \in l_2 : x_1 < x_2$. בגלל זרות האינטרוולים, היחס מוגדר באופן מלא על כל זוגות תתי הרשימות.
את סדר הרשימות נשמור בתוך עלי עץ בינארי מושלם כמו האלגוריתם שראינו בשיעור, נשמור על צפיפות $\frac{1}{T^i}$ בכל תת עץ וכו'.

<u>$order(x, y)$:</u> מחלקים ל2 מקרים:
1. $l(x) = l(y)$: אז נשווה את $label(x), label(y)$ - אם $label(y) > label(x)$ אז $y > x$, אחרת הפוך.
2. $l(x) \neq l(y)$: אז נשווה את אינדקס העלים שבהם ממקומים $l(x), l(y)$. אם $l(x)$ ממוקם בעלה ימני יותר אז $l(x) > l(y)$ ולכן $x > y$, אחרת הפוך.

<u>$insert(x, y)$:</u> ראשית מכניסים את y ל$l(x)$ ונותנים לו תגית בערך הממוצע בין $label(x), label(s(x))$ (אם $s(x)$ בתוך $l(x)$) או $label(x) + 1$ (אם $s(x)$ לא בתוך $l(x)$).
כעת נבדוק את אורך $l(x) = l(y)$, נסמן אותו בσ. אם $\sigma \geq 2k$, אז נפצל את $l(x)$ ל$\tau = \left\lceil \frac{\sigma}{k} \right\rceil$ רשימות בגודל לכל היותר k כ"א (בצורה טריוויאלית – נעבור על $l(x)$ ו"נגזור" בכל פעם שעברנו k איברים, ואז נחשב מחדש תגיות בכל תת רשימה). נוציא את $l(x)$ מעץ הרשימות ונוסיף את τ הרשימות החדשות במקומה ע"פ הסדר.

<u>$delete(x)$:</u> ראשית נוציא את x מהרשימה $l(x)$. אם $l(x)$ ריקה, נמחק אותה מהעץ T. נבדוק את הצפיפות בשורש T ונקטין אותו במידת הצורך (כמו שעשינו באלגוריתם המקורי).

נכונות ברורה מתכונות המבנה שתיארנו.

<u>סיבוכיות:</u> נראה שהסיבוכיות עם $k = \log n$ היא $O(1)$ *worst-case* עבור *order*, ו$O(1)$ *amortized* עבור insert, delete.

order: במקרה הראשון ($l(x) = l(y)$) מבצעים השוואה בין תגיות ממוצעות בתוך הרשימה. ראינו שמספר הביטים שנדרשים לייצוג התגיות לינארי בגודל הרשימה. גודל הרשימה חסום ע"י $k = \log n$, ולכן מספר הביטים שנדרשים הוא $\log n$, וההשוואה בזמן $O(1)$. במקרה השני ($l(x) \neq l(y)$)

מבצעים השוואה בין מיקומי עלים בעץ המושלם, שבו $M = O\left(\frac{n}{k}\right)$ עלים. עבור $k = \log n$, $O\left(\frac{n}{k}\right) = O\left(\frac{n}{\log n}\right) = O(n)$, ולכן ההשוואה מתבצעת בזמן $O(1)$.

insert: כמו במבנה שתואר בשיעור, גם כאן הרוב המוחלט של הפעולות הן זולות ($O(1)$), חלק קטן קצת יקרות ($O(\log n)$ עבור פיצול), וחלקן יקרות מאוד (עבור relabel).

- פיצול רשימה l לתתי רשימות :עלות הפיצול היא$O(k)$. כשהרשימה l נוצרה, היו בה לכל היותר k איברים, וכעת יש בה $2k$ ולכן מפצלים אותה. בין לבין התווספו לפחות k איברים לl. נחייב $O(1)$ עבור הפיצול לכל איבר שנוסף לl מרגע שנוצרה ועד שפיצלנו אותה.
- relabel: נניח מבצעים relabel בצומת o ברמה i בעץ, שצפיפותו לכל היותר $\frac{1}{Ti}$. עלות הrelabel היא $O\left(\frac{2^i}{T^i}\right)$ (מספר העלים המלאים בתת העץ). אחרי הrelabel, הצפיפות בכל אחד מהבנים של o היא לכל היותר $\frac{1}{T^i}$. עד הrelabel הבא, לאחד מהבנים נוספו לפחות $2^{i-1} \cdot \left(\frac{1}{T^{i-1}} - \frac{1}{T^i}\right)$ עלים מלאים ברשימות חדשות. לכל רשימה חדשה כזו, נוספו לפחות k איברים שהובילו לייצור שלה. בסה"כ בין שני relabelים עוקבים על o התווספו $\left(\frac{1}{T^{i-1}} - \frac{1}{T^i}\right) \cdot 2^{i-1} \cdot k$ איברים, נחלק את עלות הrelabel ביניהם, ונקבל עלות לכל איבר:

$$\frac{\frac{2^i}{T^i}}{\left(\frac{1}{T^{i-1}} - \frac{1}{T^i}\right) \cdot 2^{i-1} \cdot k} = \frac{2^i}{T^i} \cdot T^i \cdot \frac{1}{T-1} \cdot \frac{1}{2^{i-1}} \cdot \frac{1}{k} = \frac{2}{k(T-1)}$$

כמו באנליזה המקורית, איבר x משלם על $\log M = O\left(\log \frac{n}{k}\right) = O(\log n)$ הורים של $l(x)$ בעץ, ולכן העלות לכל איבר היא $\frac{2}{k(T-1)} \cdot \log n$. אם $k = \log n$ אז העלות היא $\frac{2}{T-1}$ כלומר קבועה.

בסה"כ כל צומת x משלם $O(1)$ עבור ההכנסה שלו לרשימה $l(x)$, $O(1)$ על הפיצול העתידי הראשון של $l(x)$, ו$O(1)$ עבור כל הrelabelים העתידיים הראשונים של ההורים של $l(x)$ בעץ.

***delete*:** האנליזה דומה: אם $l(x)$ לאחר המחיקה לא ריקה אז הפעולה עולה $O(1)$. אחרת צריך למחוק את $l(x)$ מהעץ. אם הצפיפות בשורש לא נמוכה מדי (מספר העלים המלאים לא קטן בחצי) אז עדיין $O(1)$. אחרת מקטינים את העץ פי 2 ומבצעים *relabel* בשורש. עלות הקטנת העץ וה*relabel* היא $O\left(\frac{n}{k}\right) = O(n)$, ובין כל 2 פעולות הקטנה מתבצעות $O(n)$ פעולות מחיקה (לאו דווקא $O(nk)$ כי גודל רשימות יכול להיות קטן בהרבה מk). אז מספיק לחייב עוד $O(1)$ עבור כל איבר שנמחק כדי שנוכל לשלם על הקטנה עתידית ראשונה של העץ.

שאלה 3

מאחסנים את קודקודי V בתוך מערך A בגודל n, כל קודקוד v זוכר את האינדקס שלו $i(v)$ במערך.
נשמור את האינווריאנטה הבאה: לכל $u,v \in V$, אם $(u,v) \in E$ אז $i(u) < i(v)$, כלומר הi-ים מהווים סדר טופולוגי על הקודקודים.

בתחילת הריצה אין קשתות ולכן האינווריאנטה מתקיימת באופן ריק.

נניח שהוספנו קשת חדשה (u,v).
אם $i(u) < i(v)$ אין צורך לעשות כלום, ואין מעגל חדש.
אחרת, $i(v) < i(u)$. נתחיל בלבדוק אם יש מעגל: נריץ DFS החל מv, נכניס קשתות (a,b) לתור רק אם $i(b) < i(u)$, ונדווח על מעגל אם נגיע לu.
נניח שלא מצאנו מעגל. כעת יש צורך לתקן את המיקומים בשביל האינווריאנטה.
נריץ שני DFSים בשילוב עם בניית סדר לקסיקוגרפי:

1. החל מv קדימה, לכל קשת (a,b) נכניס את b לתור רק אם $i(b) < i(u)$, נסמן את הסדר הלקסיקוגרפי המתקבל D_v (v ראשון)
2. החל מu אחורה, לכל קשת (a,b) נכניס את a לתור רק אם $i(v) < i(a)$, נסמן את הסדר הלקסיקוגרפי המתקבל D_u (u אחרון).

נסמן $D = D_u \circ D_v = u_k, u_{k-1}, \dots, u_1, u, v, v_1, \dots, v_l$. D הוא סדר טופולוגי של הקודקודים בD לאחר הכנסת הקשת החדשה (u,v). בD יש $k+l+2$ קודקודים שמתפרשים על פני אינדקסים שונים בA. נרצה להקצות מחדש את כל האינדקסים הנ"ל, כך שסדר הקודקודים בD יהיה זהה לסדר שלהם בA.
נבצע merge-sort בין D_u, D_v ע"פ האינדקסים i, נסמן את הרשימה המתקבלת F. כעת נעבור על הקודקודים בD לפי הסדר, לקודקוד $D[j]$ ניתן את האינדקס החדש $F[j]$.

```
insert(u, v):
    if i(u) < i(v):
        return // no cycle
    else: // i(v) < i(u)
        DFS from v to u, relaxing only edges strictly between (u, v); if u is found,
        report a cycle

        D_v := forward DFS from v, relaxing only edges strictly between (u, v)
        D_u := backwards DFS from u, relaxing only …
        D := D_u ∘ D_v
        F := [i(v) for v in merge_sort(D_u, D_v)] // in-order indices in D
        for j = 1 to |F|:
            i(D[j]) = F[j] // assign new index
```

<u>טענה:</u> בסיום $insert$, האינווריאנטה נשמרת.
<u>הוכחה:</u> נניח שהאינווריאנטה נשמרת לפני, ומכניסים קשת (u,v). נסמן i האינדקסים לפני ההכנסה, i' אחרי. אם הקשת $i(u) < i(v)$ אז $\forall v \in V: i'(v) = i(v)$ והאינווריאנטה בבירור נשמרת.
נניח $i(v) < i(u)$, תהי קשת $(a,b) \in E$. נפריד למקרים:

- $a,b \notin D$: אז $i'(a) = i(a), i'(b) = i(b)$ והאינווריאנטה נשמרת על פי הנחת האינדוקציה.
- $a,b \in D$: נסמן d_a = האינדקס של a בD, d_b האינדקס של b בD. D מסודרת טופולוגית ולכן $d_a < d_b$. האינדקסים בF מונוטוניים עולים ממש. בלולאת הfor אנו עוברים על איברים D לפי הסדר ומקצים להם אינדקסים מונוטוניים עולים מF, כלומר $j < j' \Rightarrow i'(D[j]) = F[j] < F[j'] = i'(D[j'])$, ובמקרה שלנו $d_a < d_b \to i'(a) = i'(b)$, והאינווריאנטה נשמרת.
- $a \in D, b \notin D$: אז $i(v) < i(a) < i(u) < i(b)$, a נגיש מv או u נגיש מa. בלולאת הfor, האינדקס המקסימלי שאיבר $D[j]$ מקבל הוא $i(u)$, ולכן בסיום הלולאה $i'(a) \le i(u) < i(b) = i'(b)$ והאינווריאנטה נשמרת.

- $a \notin D, b \in D$, אז $i(a) < i(v) < i(b) < i(u)$, ו$b$ נגיש מv או u נגיש מb. בלולאת הfor, האינדקס המינימלי שאיבר $D[j]$ מקבל הוא $i(v)$, ולכן בסיום הלולאה $i'(a) = i(a) < i(v) \leq i'(b)$ והאינווריאנטה נשמרת.

נכונות נובעת באופן ברור מהאינווריאנטה.
<u>סיבוכיות:</u> העלות של פעולת $insert$ עם קשת אחורית ($i(v) < i(u)$) פרופורציונלית ל$|D|$ (מספר הקודקודים שהאינדקס שלהם התחלף).
ניזכר בהגדרה של זוגות קודקודים וקשתות שהם "קשורים" (related): קודקוד v וקשת (a, b) הם "קשורים" אם יש מסלול בגרף העובר דרך הקשת (a, b) והקודקוד v (לא משנה באיזה סדר). יש לכל היותר mn זוגות קשורים בגרף. כל זוג הופך להיות קשור בשלב כלשהו בו מתווסף קשת שמחברת מסלול ביניהם, ומעתה והלאה נשאר קשור (כי לא מוחקים קשתות).
הוספה של קשת אחורית (u, v) ($i(v) < i(u)$) הופכת את u לקשור לכל הקשתות בD_v ואת v לקשור לכל הקשתות בD_u. בסה"כ נוספו $|D|$ זוגות קשורים חדשים. אז בכל ריצת האלגוריתם, יכולים להווצר לכל היותר mn זוגות קשורים, ולכן הסיבוכיות הכוללת של כל ההוספות, שהיא סה"כ $|D|$ על פני כל ההוספות, חסומה ע"י $O(mn)$.

שאלה 4

G גרף k-פריק אם לכל $G' = (V', E') \subseteq G$, אפשר לחלק את V' ל3 קבוצות A, S, B כך ש:

- $|S| \leq k$
- לכל $a \in A, b \in B$ מתקיים $(a,b), (b,a) \notin E$
- אם $|E'| = m'$, אז $|\{(a,b) \in E : a \in A \text{ or } b \in A\}| \leq \frac{2m'}{3}$, ובצורה זהה ל$B$.

נקרא לחלוקה כזאת לA, S, B חלוקה חוקית אם היא עונה על 3 הקריטריונים.

נניח שעשינו m הכנסות וקיבלנו G_m שהוא k-פריק. ניקח חלוקה חוקית כלשהי A, S, B של G_m.
<u>למה:</u> מספר הזוגות (s, e) עם $s \in S$ קודקוד וe קשת (כלשהי) הקשורים זה לזה (במובן שראינו בשיעור – שיש מסלול כלשהו בגרף שמכיל את s, e) הוא לכל היותר km.
<u>הסבר:</u> $k \leq |S|$, כל זוג של קודקוד וקשת יכול להיות related לכל היותר פעם אחת, יש km זוגות כאלה.

<u>מוטיבציה:</u> נסתכל "רטרואקטיבית" על הכנסות הקשתות האחוריות שביצענו. נשאף "לחייב" כמה שיותר מההכנסות האלה על זוגות קשורים $(s, e) \in S \times E$, שהם לכל היותר km. יהיו הכנסות שלא נוכל לחייב לזוגות הנ"ל – ובהכנסות האלו נטפל בנפרד ע"י הפעלה רטרואקטיבית של הפירוק על תתי הקבוצות A, B.

לפי הגדרת גרף k-פריק, אין אף קשת $e \in A \times B$ או $e \in B \times A$ מבין m הקשתות שהכנסנו.
הקשתות האפשריות הן:

1. קשתות הנוגעות בקודקוד מS: $A \times S, S \times A, B \times S, S \times B, S \times S$, שייקראו **קשתות מסוג 1.**
2. קשתות שלא נוגעות בקודקוד מS: $A \times A, B \times B$, שייקראו **קשתות מסוג 2.**

נניח שבשלב כלשהו הכנסנו קשת $e = (u, v)$ אחורית, במובן שברגע זה $i(v) < i(u)$ (i האינדקס בסדר הלקסיקוגרפי של הקודקודים שהאלגוריתם שומר ברגע ההכנסה). כעת ביצענו חיפוש דו-כיווני מv ומu, וקיבלנו זוג קבוצות של קשתות שנסרקו: קבוצה אחת קדימה מv, נסמן אותה D_v, והשנייה אחורה מu, נסמן אותה D_u. העבודה המושקעת בהכנסה זו היא $O\big((|D_v| + |D_u|) \cdot \log n\big)$.
האלגוריתם פועל בצורה שבה $|D_u| = |D_v|$ (עד כדי קבוע), ולכן מספיק לחסום את אחד מהם.

אם e קשת מסוג 1, אז לה"כ $v = s \in S$ (המקרה השני סימטרי). אז כל הקשתות בD_u לא היו קשורות לs לפני ההכנסה, ועכשיו הן כן. לכן, את כל ההכנסות של הקשתות מסוג 1 אפשר לזקוף לחובת זוגות קשורים $S \times E$ בניתוח הסיבוכיות.
אם e קשת מסוג 2, נפריד למקרים:
אם קיים קודקוד $s \in S$ שנמצא בD_u (כלומר, שהDFS מu עבר דרך s), אז כל הקשתות מD_v לא היו קשורות לs לפני ההכנסה ועכשיו הן כן. בצורה סימטרית אם s נמצא בD_v. אז גם את המקרה הזה ניתן לזקוף לחובת זוגות קשורים $S \times V$.

נותרנו עם קשתות מסוג 2 (כלומר, $A \times A$ או $B \times B$), שבזמן ההכנסה אף קודקוד $s \in S$ לא הופיע בעץ הDFS מקצותיהן. נסתכל על המקרה של $A \times A$ (השני סימטרי). נניח הקשת $e = (a_1, a_2) \in A \times A$. לכל קודקוד $a \in A$, כל השכנים הם או מA או מS (לעולם לא מB). אז כאשר האלגוריתם מבצע חיפוש DFS קדמי החל מa_2, בכל קשת (a, v) או ש$v \in A$ או ש$v \in S$. אם $v \in S$ אז חוזרים למקרה הראשון (יש קודקוד $s \in S$ ב$D_u \cup D_v$). אחרת, כל הקשתות בעץ הDFS החל מa_2 קדימה הן פנימיות לA. בצורה דומה, כל הקשתות בעץ הDFS אחורה החל מa_1 הן פנימיות לA. אז $D_u \cup D_v$ מכילים רק קשתות פנימיות לA, וכל הקודקודים שישנו את מיקומם בסדר הטופולוגי הם קודקודים בתוך A.
נגדיר את הגרף G_A, בו קבוצת הקודקודים היא A וקבוצת הקשתות היא כל הקשתות בG שהן פנימיות לA. בהכנסת הקשת הפנימית (a_1, a_2) כל הקשתות שעברנו בהם וכל הקודקודים ששינו את מקומם הם בתוך G_A, ולכן ניתן "לצמצם" את ההסתכלות שלנו לתת הגרף G_A לצורך ניתוח הסיבוכיות. G_A הוא תת גרף של G_m, וכזה, הוא k-פריק בעצמו. בנוסף, מספר הקשתות בG_A הוא לכל היותר $\frac{2m}{3}$ (לפי הגדרת k-פריק). נפעיל ברקורסיה את אותו ניתוח בדיוק על G_A: נפרק אותו

לA_1, S_1, B_1, נטען שמספר הזוגות $(s, e) \in S_1 \times E(G_A)$ הוא לכל היותר $k \cdot \frac{2m}{3}$, ונחסום כך את סיבוכיות הכנסות הקשתות מסוג 1 בתוך G_A. נמשיך ונפרק לתתי גרפים עד שנגיע לגרף "קטן מספיק", נניח לגרף G' שבו לכל היותר $\sqrt{n}$ קשתות, ובו הסיבוכיות הכוללת של ההכנסות תהיה $O(n)$ (באמצעות ניתוח טריוויאלי של זוגות קשורים ללא הפרדה לתתי קבוצות). את אותה הפרדה נבצע גם על תתי הקבוצות שבB.

מתקבל חסם רקורסיבי על סיבוכיות מספר הקשתות ב$D_u \cup D_v$ לאורך תתי העצים.

נסמן $\psi(m, n)$ החסם על מספר הקשתות ב$D_u \cup D_v$ על הגרף המלא, שבו n קודקודים וm קשתות.

לפי הניתוח הנ"ל:

$$\psi\left(\sqrt{n}, n\right) = O(n)$$

$$\psi(m, n) = O(km) + 2\psi\left(\frac{2m}{3}, n\right) \left(\sqrt{n} < m \leq n^2\right)$$

באופן כללי הנוסחא השנייה, אם m צריך לרדת עד לערך קבוע, נפתחת ל$\psi(m, n) = O(km^c)$ עם $c > 1$. בזכות העובדה שאנחנו צריכים לרדת רק עד $\sqrt{n} \geq m^{\frac{1}{4}}$, היחס הנ"ל נפתח ל:

$$\psi(m, n) = O(km \log m) = O(km \log n)$$

(תחת ההנחה ש$m > n$).

לחסם הנ"ל נוסף פאקטור אחד של $O(\log n)$ בעבור תחזוקת D_u, D_v בערימות ע"י האלגוריתם, ומקבלים $O(km \log^2 n)$.