פתרון תרגיל מספר 3 בקורס
נושאים מתקדמים באלגוריתמים ומבני נתונים

10/10

① נגדיר semi-splaying כתהליך כמו splaying בין $p(p(x))$ ל-$p(x)$, וגם בין x ל-$p(x)$, בו רק את הצלע בין $p(p(x))$ ל-$p(x)$, נניח.

נוכיח כי ה- access lema מתקיימת גם אם מתלפים ה- zig-zig ב semi splaying:

נזכר ב access lema:

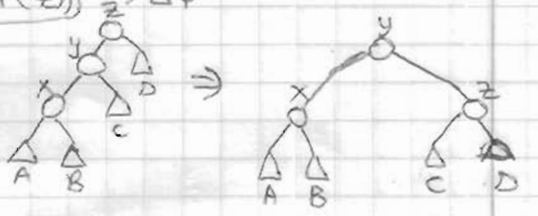
נגדיר:
$$S(x)=\sum_{u\in \text{subtree of } x} w(u)$$
$$r(x)=\log(S(x))$$

אזי ה- am.t לביצוע פעולה splay של עץ שורשו t הקודקוד x הוא לכל היותר $3(r(t)-r(x))+1=O(\log(\frac{s(t)}{s(x)}))$

הוכחה:

תחילה, נראה כי אכן ה- a.t של semi.s הוא לכל היותר $3(r'(y)-r(x))$ (כאשר s, s', r, r' הם הגודל ודרגה של x בדיוק לפני ואחרי הצעד.

מתבצעת רוטציה אחת ולכן ה- a.t הוא:
$$am.t(semi.s)=\overbrace{1}^{ac.t}+\underbrace{(r'(y)+r'(z))-(r(y)+r(z))}_{\Delta\Phi}$$



באת נשים שתת העץ של x לא השתנה, ולא נעשה השתנו רק תתי העצים של y ושל z, ושביצענו רק סיבוב 1.
נשים לב כי: $r'(y)=r(z)$
שכן ישנם אותם קודקודים בשני תתי עצים אלו, ולכן:

$$am.t(semi.s)=1+r'(z)-r(y)\le 2+r'(z)-r(y)\le 2+r'(z)-r(x)$$

שכן: $r(y)=\log(w(y)+s(x)+s(z))\ge\log(s(x))=r(x)$, $r(y)\ge r(x) \Rightarrow -r(y)\le -r(x)$

נראה כי: $-2\ge -2r'(y)+r'(z)+r(x)$
ונקבל כי: $2+r'(z)-r(x)\le 2(r'(y)-r(x))$

ואכן:
$$r'(z)+r(x)-2r'(y)=\log\left(\frac{s'(z)}{s'(y)}\right)+\log\left(\frac{s(x)}{s'(y)}\right)$$

ומתקיים: $s'(y)=\overset{s'(x)}{s(x)}+s(z)+w(y)\Rightarrow s'(z)+s(x)\le s'(y)$
ובמונח $s'(y)>0$ ולכן: $\frac{s'(z)}{s'(y)}+\frac{s(x)}{s'(y)}\le 1$
וכן: $\frac{s'(z)}{s'(y)},\frac{s(x)}{s'(y)}>0$ (שכן אלו גדלים אי-שליליים לפי הגדרת w).

וכפי שאמרנו בכיתה על פי תכונות הקמירות של פונקצית log מתקיים כי עבור $0<x,y$, $x+y\le 1$, $\log x+\log y$ מתקבל בערך -2 עבור $x=y=\frac{1}{2}$, ולכן:
$$-2r'(y)+r'(z)+r(x)=\log\left(\frac{s'(z)}{s'(y)}\right)+\log\left(\frac{s(x)}{s'(y)}\right)\le -2$$

ולכן: $am.t(semi.s)\le 2(r'(y)-r(x))$

ה- am.time עבור צעד zig וצעד zig-zag נשאר אותו הדבר,

ומסכמת ה- amtime של כל שלבי ה-splaying (כאשר נשים לב

בסכימה כי לבסוף צעד semi-splaying משאיר את ה-splaying על הקודקוד

y (ולא על הקודקוד x), נקבל ~~כצעדים~~ (בדומה לכיתה) סכום

טלסקופי המוביל לכל היותר $2(r(t)-r(x)) = \cancel{3}\,2(r'(x)-r(x))$

(ניזכר כי ~~בצעד~~ בכיתה קיבלנו כי:

$$am.t(zig) \leq 1 + r'(x) - r(x) \leq 1 + 2(r'(x)-r(x))$$

$$am.t(zig\text{-}zag) \leq 2(r'(x)-r(x))$$

(ובסיכום מתקיים: $r(t) \geq r(x) \Leftarrow am.time \leq 2(r(t)-r(x)) \leq 3(r(t)-r(x))$.

(2) נוכיח כי ה-access lemma תקפה ל-splaying מסוג זה:

$\frac{10}{10}$

תחילה נדגיש כי כמו ההוכחה בכיתה, גם כאן ה-am.time של ה-top down splay הוא סכום ה-am.time של הצעדים של הצעד, שהוא א"ר לסקופי (שכן בסוף פעולה על x, נשיג שהטופס-ששמש x הינו האיבר שממנו משיכים את ה-splay), ונקבל:

$$am.time(\text{top down splay}) \le 2(r(t)-r(x)) = O\left(\log\left(\frac{s(t)}{s(x)}\right)\right)$$

נראה עתה כי:

$$am.time(zig\text{-}zig) \le 2(r(u)-r(v))$$

וגם:

$$am.time(zig\text{-}zag) \le 2(r(u)-r(v))$$

$$am.time(\text{top down zig zig}) = \underbrace{1+r'(x)+r'(y)+r'(z)-r(x)-r(y)-r(z)}_{\Delta\Phi(T)}$$

$$\overset{r'(y)=r(z)}{=} 1+r'(x)+r'(z)-r(x)-r(y)$$

$\log a+\log b \le 2\log\frac{(a+b)}{2}$ שכן

עבור $a,b>0$ כפי שראינו ב-1

$$\le 1+\underbrace{2\log(s'(x)+s'(z))-2}_{\ge \log(s'(x))+\log(s'(z))}-r(x)-r(y)$$

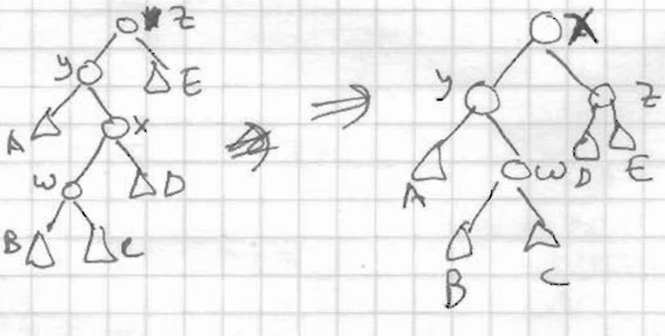
כיוון x,y,z קשתו 1

$$\overset{s'(y)=s'(x)+s'(z)+1}{\le} 1+2\log(s'(y))-r(x)-r(y) = 2r'(y)-r(x)-r(y) =$$

$$= 2r(z)-r(x)-r(y) \le 2(r(z)-r'(x)) = 2(r(u)-r(v))$$

$(r'(x)=r(x))$

:zig zag



$$am.time(\text{top down zig zag}) = 2+\underbrace{r'(x)+r'(y)+r'(z)-r(x)-r(y)-r(z)}_{\Delta\Phi(T)}$$

$$\overset{r'(x)=r(z)}{=} 2+r'(y)+r'(z)-r(x)-r(y) = 2+\log(s'(y))+\log(s'(x))-r(x)-r(y)$$

שוב כי $\log a+\log b \le 2\log(a+b)-2$ כמו ב-1 עבור $a,b>0$

$$\le 2+2\log(\underbrace{s'(y)+s'(z)}_{})-2-r(x)-r(y)$$

$s(x)=s'(y)+s'(z)+1$

$$\le 2r'(x)-r(x)-r(y) \overset{r'(x)=r(z)}{=} 2r(z)-r(x)-r(y) \overset{r(w)\le r(x),r(y)}{\le} 2(r(z)-r(w))$$

$r(\omega)=r'(\omega)$

$\overset{\downarrow}{=} 2(r(z)-r'(\omega)=2(r(u)-r'(v))$

(3) נראה איך להרחיב את dynamic trees שתומך בפעולה evert(v) ב-O(log(n)) זמן ריבה, ללא פגיעה בפעולות האחרות.

10/10

evert(v) - שינוי העץ האמיתי המכיל את v, כך ש-v היא ה-root של העץ לאחר הפעולה.

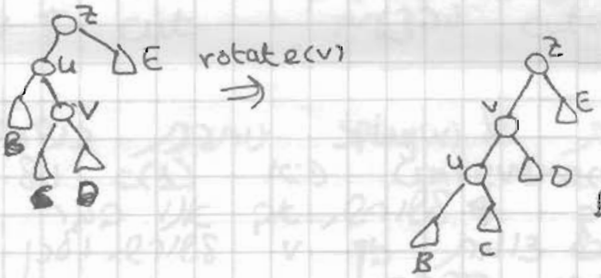
לשם כך נוסיף לכל node בעץ הוירטואלי reverse bit שנסמן b(v). נסמן ב- B(v) את הזוגיות של הסכום של ה-reverse bits בדרך מ-v ל- solid root(v).

נגדיר כי אם u הוא solid child של v אז:
אם: u בן שמאלי של v ו- B(v)=0,
או u בן ימני של v ו- B(v)=1,
אז: u בן שמאלי אמיתי של v.
אחרת: u בן ימני אמיתי של v.

עתה נראה תחילה כיצד ניתן לבצע עתה rotate(v) ו- splice(v):
תנאי מוקדם הנוסף לביצוע rotate(v) הוא שאם u=p(v) ב- solid tree, אז: b(v)=b(u)=0.
splice(v) דורש כי v יהיה dash child של solid root u (כפי שהגדרנו בכיתה), אך הנוסף כי b(u)=0

אנו דורשים תנאי אלו על ה- reverse bit על מנת שהפעולות לא ישנו את הזוגיות של צמתי העצים של v ו-u
ה-nodes B(ω)

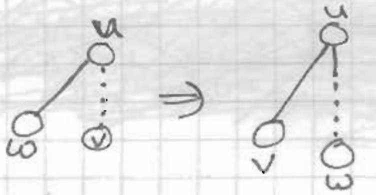
ב- rotate(v):



ולכל ω∈D, מתקיים:
זוגיות הסכום לפני פחות b(u)
$B'(\omega) = B(\omega) - b(u)$
משמע אם $b(u)=0$ אז $B'(\omega)=B(\omega)$

ולכל ω∈B מתקיים
$B'(\omega) = B(\omega)+b(v)$
(משמע זוגיות הסכום לפני ועוד b(v), ולכן $b(v)=0$ אז $B'(\omega)=B(\omega)$
כל השאר נותרים אותו הדבר.
ה- B(ω)

ב- splice(v)



לכל t בתת העץ של v
הזוגיות B'(t) הופכת להיות הזוגיות של הסכום קודם ועוד b(u), ולכל r בתת העץ של ω
הזוגיות B'(r) הופכת להיות הזוגיות של הסכום קודם פחות b(u)
(על פי הגדרת הזוגיות), ולכן b(u)=0 את הזוגיות נותנת אותו הדבר.

על כן הפעולות splice(v) ו- rotate(v) שומרות על מבנה ~~העץ המשמעותי~~
הבנים השמאליים והימניים האמיתיים בעץ הוירטואלי,
ומכאן שאינן פוגעות במבנה העץ האמיתי ותחת תנאים אלו.

נגדיר עתה: unreverse(v):

אם $b(v)=1$:
נשנה $b(v)=0$, ונהפוך בין הבן השמאלי ה־solid לימני של v.
נהפוך את $b(l)$ ו־$b(r)$ של הבנים ה־solid הימני והשמאלי של v.

הנכונות של תתי העצים של l ו־r (הבנים ה־solid השמאלי והימני של v בהתאמה) נותרת אותו הדבר (שכן ההיפוך של $b(v)$ "מתקזז" עם ההיפוך של $b(l)$ ו־$b(r)$).
הנכונות של l ו־r עצמם מתהפכת, אך גם המיקום הפיזי שלהם מתהפך, ולכן הבן הימני האמיתי נותר הבן הימני האמיתי וכנ"ל גם השמאלי, ומבנה העץ נשמר.

עתה בביצוע פעולת ה־splay(v) נבצע עתה unreverse(u) על כל ה־node u שבדרך בין v לשורש l של ה־solid tree המכיל את v שבראש העץ הוירטואלי, ואז נבצע את splay(v) כרגיל.
לכל node u בדרך מתקיים עתה $b(u)=0$, ולכן ניתן לבצע splice(u) (אם התנאי השני מתקיים כמובן) ו־splay בכל solid subtree כרגיל.

משום ש־splay וכל הפעולות על $\Delta cost$ ו־Δmin שומרות על נכונותן (נשים לב עתה כי $\Delta cost(right(v))$ הוא $\Delta cost$ של הבן הימני האמיתי של v וכנ"ל כמובן ל־Δmin ולבן השמאלי האמיתי) כל יתר הפעולות שומרות על נכונותן.
נשים לב גם כי ב־cut(v) מבצעים cut בין v לבן הימני האמיתי של v.

נשים לב כי הסיבוכיות של splay(v) נותרה כשהיתה.
שכן הסיבוכיות של unreverse(v) היא $O(1)$ ולכן נוסף $O(1)$ לכל צומת בדרך לשורש, אך אנו כבר מבצעים פעולות בלפחות $O(1)$ לכל צומת בין v לשורש, ולכן הסיבוכיות של splay(v) נותרת אותו הדבר.
כמובן שגם הסיבוכיות של יתר הפעולות נותרת אותו הדבר, שכן לא שינינו אותן.

~~[illegible]~~

עתה נגדיר:

:evert(v)

splay(v)

אם ל-v יש בן ימני w אז:

disconnect(w)

~~נתלה את~~

נהפוך את b(w)

אם ל-v אין בן שמאלי אז:

נהפוך את w לבן שמאלי של v

אחרת:

נהפוך את w לבן מקומו של v, וננתק את ה...

אחרת v כבר ה-root של העץ האמיתי.

נראה כי evert(v) מבצע את שהוא אמור לבצע:

לאחר ביצוע splay(v), v הוא השורש של העץ הוירטואלי.

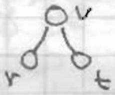
לכן כל תת העץ הימני הסולידי שלו הוא המסלול בינו לבין שורש העץ האמיתי.

נשים לב כי היפוך b(v) גורם ל-inorder של תת העץ ששורשו v להתהפך.

ניתן להוכיח באינדוקציה על גובה העץ:

$h=2$

inorder: r,v,t

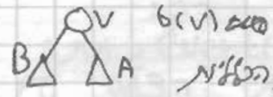


(נניח בלי הגבלת הכלליות כי b(v)=0)

$b(v)=1-b(v) \Rightarrow$ t בן שמאלי אמיתי של v, r בן ימני אמיתי של v

ולכן inorder: t,v,r

עבור h+1:



(נניח בלי הגבלת הכלליות כי B(v)=0)

inorder: $b_1,\dots,b_l,v,a_1,\dots,a_k$

$b(v)=1-b(v) \Rightarrow$ A בן שמאלי אמיתי של v, B בן ימני אמיתי של v (שכן היפוך של b(v) גורם להיפוך B(v) ומכאן להיפוך בין הבן הימני האמיתי לשמאלי האמיתי)

ולכן

$$\text{inorder}=\underbrace{a_k,\dots,a_1}_{},\ v,\ \underbrace{b_l,\dots,b_1}_{}$$

לפי הנחת האינדוקציה

לכן בפעולת היפוך b(v) מתהפך סדר האיברים בין v לשורש האמיתי הישן. את מיקום v מתקנים כשמעדכנים את המסלול תחת v על ידי הפיכת w לבן שמאלי, או אפס, אם קיים כבר בן שמאלי.

וקיבלנו כי e_{vert} פועל כמצופה ☺

זמן הריצה של evert: $O(1)+splay = O(\log n)$

(4) עבור n nodes אלו נגדיר את המבנה הבא:

נשמור מונה t הגדל בכל insert של קשת, ולמעשה יתאר את זמן הכנסת הקשת ויהווה את ה- cost של הקשת. 10

המבנה שלנו יהיה יער של עצים פורשים מקסימליים (dynamic trees) של רכיבי הקשירות.

בנוסף נשמור FIFO של הקשתות המוכנסות ושקלן, נשים לב כי המשקל של הקשתות ב-FIFO ממוין מהקטן לגדול, ושתמיד נוציא את הקשת עם המשקל הקטן ביותר (שהיא גם ה- oldest), משום שהיא הראשונה מכל הקשתות ב- FIFO שהוכנסה.

עתה נראה את המימוש של כל פעולה במבנה זה, ושהיא משמרת את המבנה:

<u>insert(u,v):</u>

כמובן כי תחילה נגדיל את t באחד, וניתן לקשת (u,v) את המתיר t, ונכניס את הקשת (ומשקלה (t,v,u)) לתוך ה- FIFO. עתה נרצה להכניס ליער את הקשת (v,u).

נבצע connected(u,v) ונבדוק אם u ו-v מחוברים בגרף. אם הם לא מחוברים ניתן פשוט לבצע link(u,v,t) באופן הדינאמי, קודם היו לנו שני עצים פורשים מקסימליים, ועכשיו נקבל עץ פורש אחד שהוא בהכרח מקסימלי (שכן היא הקשת הכבדה היחידה האפשרית - קודם העצים לא היו מחוברים).

אם הם כן מחוברים, נשים לב שהכנסת הקשת תגרום לסגירת מעגל, ולכן צריך קודם להוציא את הקשת בעלת המשקל הנמוך ביותר (שהיא גם ה"זקנה" ביותר) במסלול בין u ו- v מהעץ.

נעשה זאת כך:

נבצע evert(u), עתה u הוא שורש העץ, ונבצע mincost(v) המתליר את הצלע עם המשקל הקטן ביותר בדרך בין v לשורש העץ (u), ונבצע cut(w) כש-w הקודקוד הקרוב יותר לשורש שבקצה הצלע שהתקבלה (mincost(v)=(w,z)).

לאחר מכן ניתן להכניס את הצלע (u,v) לעץ בלי שנוצר מעגל. העץ עדיין פורש (שכן תיברנו תמורה בין (u,v)), והוא בהכרח מקסימלי, שכן הוצאנו את הקשת בעלת המשקל הקטן ביותר בעץ, והקשת שהכנסנו בהכרח בעלת משקל הגדול ממשקל כל הקשתות בעץ (על פי הגדרת המשקל הניתן לקשת), ולכן בסוף פעולת ה- insert אנו נשארים עם עץ פורש מקסימלי.

הפעולות שאנו מבצעים הן:

הגדלת t והכנסת הקשת ל-FIFO - $O(1)$

connect(u,v) - נראה בהמשך כי נעשה ב- $O(\log n)$

evert(u), mincost(v), cut(w), link(u,v,t) - פעולות dynamic trees שהוכחנו כי ה- am.time שלהן הוא $O(\log n)$

ולכן קיבלנו כי הזמן של insert(u,v) הוא $O(\log n)$ וכי הוא אכן משמר את מבנה היער.

Connected(u,v):

נרצה לענות האם קיימת דרך בין u ל-v בגרף הנוכחי.
לשם כך נבצע ב-dynamic tree - findroot(u) ו-findroot(v),
אם יש להם את אותו השורש, משמע ~~את~~ הם נמצאים באותו עץ פורש, ולכן באותו רכיב קשירות, ונחזיר true.
אחרת הם נמצאים בעצים פורשים נפרדים ונחזיר false (עצים פורשים נפרדים, משמע רכ"ב קשירות נפרד).
כמובן שפעולות אלו לא פוגעות במבנה.

ה-am.time של findroot(v) הוא logn

delete-oldest(u,v):

נרצה למחוק את הקשת הישנה ביותר בגרף.

תחילה נמצא מה הקשת הישנה ביותר בגרף ע"י ביצוע pop של ה-FIFO (כפי שראינו בהתחלה היא הראשונה מבין הקשתות ב-FIFO).
עתה נרצה לבדוק אם הקשת שאנו מוחקים שייכת לעץ דינאמי ~~[illegible]~~ זו או לא. אם היא לא שייכת, ניתן למחוק את הקשת מהגרף, אך זה אומר שמחקנו אותה מהיער כאשר הכנסנו קשת אחרת (לכן הדרך היחידה בה מוחקים קשת לפני שהיא הכי זקנה), ולכן היא לא ביער ~~[illegible]~~
שכן ~~[illegible]~~ יש מסלול אחר בין u ו-v ולכן לא צריך לעשות כלום.
אם הקשת כן קיימת בעץ דינאמי נבצע ~~[illegible]~~ לבצע cut(v), שכן מחיקת הקשר תפריד בין u ו-v ~~[illegible]~~ ותחלק לשני רכיבי קשירות שונים. זאת משום שאין קשת אחרת ~~[illegible]~~ בגרף המתחברת בין u ו-v, שכן אנו עדיין פורשים מקסימלים ואילו זאת קשת מינימאלית בכל העץ, ולכן אם הייתה קשת אחרת ה-cost שלה היה גדול יותר היא הייתה בעץ הפורש המקסימאלי במקומה זו. עתה נקבל כי העץ מתפרק לשני עצים פורשים מקסימאליים.

מציאת הקשת (u,v) בעץ דינאמי אם היא קיימת:
נבצע evert(u) ואז mincost(v), ~~משום~~ שבביצוע evert, u הוא עתה הקודקוד, ו-mincost(v) מחזיר את הקשת המינימאלית בין v ל-u (הקודקוד).
~~אם~~ הקשת זו לא יכולה להיות במחיר הקטן מהמחיר של הקשת שברצוננו למחוק, שכן זו הקשת הכי ישנה (משמע עם המחיר הכי קטן - לפי הגדרת המחיר) שבגרף. על כן אם זו של הקשת (u,v), או שמחירה גדול ממחיר (u,v), משמע (u,v) לא ~~[illegible]~~ בעץ הפורש, שהיא לא בכלל עץ פורש.

אנו מבצעים:

הוצאה מה-FIFO - $O(1)$
evert(u), mincost(v), cut(v) על עץ דינאמי: am.time = $O(\log n)$

וסה"כ נקבל כי ה-am.time = $O(\log n)$.

5 ג נוכיח כי גירסא זו מבצעת $O(n^3)$ non.sat.pushes:

ט/ט

~~במעבר~~ ישנן $O(n^2)$ פאזות relabel, כאשר בכל מעבר יש לפחות relabel אחד, ולכן יש $O(n^2)$ מעברים.

במעבר יחיד אם מתקיימת non.sat.push של צומת, אזי הוא נהיה לא פעיל ~~access~~ ועל כן לא נבצע עוד push/relabel של הצומת ומכאן שגם לא עוד non.sat.pushes.

בנוסף, משום שהצמתים ממויינות לפי ה-dis.labels, לאחר ביצוע non.sat.push לצומת, בהכרח עברנו על ה-admissable קשתות אליה, ולכן לא יהיו access לצומת במעבר זה, ולא תתבצע ממנה דחיפה. רק לאחר relabel של צומת המוצא קשתות לצומת כן יהפכו לאדמיסבילית במעבר זה, ועל כן ישפיעו רק במעבר הבא.

על כן בכל מעבר כזה תתבצע רק עד non.sat.push אחד לכל צומת.

~~ולכן ישנם פעמים~~: ולכן ישנם סה"כ

$$\#\text{non.sat.pushes} = \underbrace{O(n^2)}_{\#\text{מעברים}} \cdot \underbrace{O(n)}_{\#\text{nodes}} \cdot \underbrace{O(1)}_{\text{non.sat.push per node per run}} = O(n^3)$$

נשמור שתי רשימות: רשימה אחת (L1) להיות ריקה ואת השנייה (L2) להכיל
בהתחלה נאתחל רשימה ב distance label של כל הצמתים הוא
את כל צמתי הגרף. בהתחלה
אפס, ולכן סדר הצמתים ברשימה לא משנה..

עתה נבצע מעבר על כל הצמתים ברשימה L1. לכל צומת:
- אם הוא אקטיבי בתחילת המעבר נבצע discharge
- נוסף אותו ל-L2 ממוין לפי ה-distance label.

(נבדוק בתחילת כל מעבר את כל הצמתים האקטיביות, ונבצע discharge רק עליהן.)

עתה כל עוד ביצענו relabel במעבר של L1, נציב את L2 ב-L1, נחזיר את L1 להיות ריקה ונחזור על המעבר של L1.

האלגוריתם עובד שכן בתחילת כל מעבר L1 מכיל את כל הצמתים בגרף, ממויינים לפי ה-dis.label שלהם, שכן לאחר כל relabel הצומת מוכנס בחזרה ממויינת לרשימה עם ה-dis.label החדש שלו.

נשים לב כי בכל מעבר אנו מתחילים (ב-discharge) מהצמתים הלא אקטיביות. בגלל שממיינים את הצמתים לפי ה-dis.label,
כפי שציינו ב-2.א - על כל ה-admible
לאחר ביצוע non saturating push לצומת כבר עברנו
השכנות הנכנסות לצומת, ולכן ל-v לא יכנס access במעבר זה.
של כן אם לא התבצע relabel במעבר, בסופו כל הצמתים לא פעילים (שכן לא נכנס access מצומת נכנסת/צומת קודם במעבר, ובסוף המעבר על כל צומת הוא לא פעיל).

זמן ריצת האלגוריתם

* כפי שראינו בכיתה מספר הפעולות relabel ~~sat.pushes~~ באלגוריתם הוא $O(n^2)$.
* הכנסה ממויינת של צומת לרשימה (הלוקחת $O(n)$) נעשית רק לאחר relabel, שכן אם לא היה relabel ניתן להכניס צומת לרשימה ב-$O(1)$ (לסוף הרשימה - שכן בהתחרת כל הצמתים שעבר ב-L2 הם עם dis.label גדול משל הצומת המוכנס, שכן הם היו ב-L1 לפניו (ו-L1 ממויינת בתחילת כל רצה), וה-dis.label שלהם יכול רק לגדול בריצה עליה בו). ולכן ההכנסות ב-$O(n)$ לוקחות סה"כ $O(n^2\cdot n)=O(n^3)$.
* בנוסף, ישנן $O(n^3)$ sat.pushes (לפי הכיתה), ו-$O(n^3)$ non.sat.pushes (לפי 2.א').
* בנוסף לפי סעיף א' אנו מבצעים $O(n^2)$ מעברים, שבמהלכם אנו מבצעים עוד $O(1)$ פעולות של כל צומת:
הכנסה ל-L2 שלא לאחר relabel - $O(1)$
בדיקה האם הצומת פעיל - $O(1)$
העברת הצומת ב-L1 ל-L2 - $O(1)$
ולכן סה"כ אנו מבצעים עוד $O(n^3)$ פעולות.

סה"כ נקבל זמן ריצה של $O(n^3)$.

ולכן: $am,t(semi,S) \le 2(r'(y)-r(x))$