נושאים מתקדמים במבני נתונים ואלגוריתמים

אלי דיין[1]

10 ביולי 2013

[1] <eliyahud@post.tau.ac.il>

תקציר

מסמך זה יביא את סיכומי השיעורים מהקורס נושאים מתקדמים במבני נתונים ואלגוריתמים,
שהועבר על ידי פרופ׳ חיים קפלן בסמסטר ב׳ בשנה״ל תשע״ג.

תוכן עניינים

1 עצי הרחבה **4**

1.1 מוטיבציה 4

1.2 ניסיון מימוש ראשון 4

1.3 ניסיון מימוש שני - הרחבה (*Splaying*) 6

1.3.1 אנליזה 6

1.3.2 שימושים 13

1.4 שימושים 14

1.4.1 דחיסה באמצעות עצי Splay 14

1.4.2 דחיסה באמצעות עצי Semi-Splay 16

1.5 פעולות על עצי Splay 16

1.5.1 `Catenate` 16

1.5.2 `Split` 17

1.5.2.1 `Split` (i,T) כאשר $i \in T$ 17

1.5.2.2 `Split` (i,T) כאשר $i \notin T$ 17

1.5.3 `Insert` 18

1.5.4 `Delete` 20

1.6 שאלות פתוחות 20

1.6.1 Dynamic Optimality Conjecture 20

1.6.2 Dynamic Finger Conjecture 21

2 עצים דינאמיים **22**

2.1 פעולות נתמכות 22

2.2 שימושים 24

2.2.1 עץ פורש מינימלי 24

2.2.2 עץ פורש מינימלי עם שני צבעים 24

2.2.3 ID Range Reporting 25

2.3 מימוש 26

2.3.1 מקרה פרטי – שרוך 26

2.3.2 המקרה הכללי 27

2.3.2.1 פעולות בעצים הוירטואליים 27

2.3.2.2 פעולות של העצים הדינאמיים 31

2.4 ניתוח 31

3 זרימה מקסימלית 34
3.1 הגדרות 34
3.1.1 זרימה 34
3.1.2 חלוקות s–t 35
3.1.3 רשת שיורית 36
3.2 האלגוריתם של Dinic 37
3.2.1 ניתוח סיבוכיות 37
3.2.2 עצים דינאמיים 38

4 זרימה מקסימלית בשיטת Push/Relabel 41
4.1 Distance Labels 41
4.2 Preflow 41
4.3 אלגוריתם Push/Relabel 42
4.3.1 נכונות 42
4.3.2 ניתוח סיבוכיות 43
4.3.3 מימוש 44
4.4 הקטנת מספר ה־Push־ים הלא מרווים 45
4.5 עצים דינאמיים 45
4.5.1 אנליזה 46
4.5.2 שיפור לאנליזה 46

5 זרימה בעלות מינימלית 48
5.1 הגדרת הבעיה 48
5.2 אלגוריתם ביטול המעגלים 50
5.2.1 ניתוח 50
5.3 פוטנציאל בצמתים 50
5.3.1 הגדרות 50
5.3.2 קריטריון האופטימליות 51
5.3.3 האלגוריתם 52
5.4 אופטימליות מקורבת 52
5.5 אלגוריתם קירוב הרצף 54
5.5.1 אנליזה 54
5.5.2 מימוש גנרי ב־$\mathcal{O}\left(n^2 \cdot m\right)$ 55
5.5.3 מימוש ב־$\mathcal{O}\left(n^3\right)$ 55
5.6 שימוש בעצים דינאמיים 56
5.6.1 ייצוג טוב יותר של L 58
5.7 גרסה פולינומית חזקה לאלגוריתם קירוב הרצף 60
5.7.1 פוטנציאלים ε־צמודים 60
5.7.2 האלגוריתם 61
5.7.3 אנליזה 61

6 האלגוריתם לזרימה מקסימלית של Goldberd & Rao 63
6.1 הקדמה 63
6.2 האלגוריתם של Goldberg & Rao 63
6.3 ניתוח 64
6.4 המשך המימוש 66

6.5 סיכום 66

7 חיבוריות בגרף דינאמי **67**
7.1 הקדמה 67
7.2 רעיון למימוש 67
7.3 מימוש יותר מדוייק 69
7.4 ניתוח 69
7.5 מימוש ה־Abstract Datatype בצורה יעילה 70
7.6 עץ פורש מינימלי בגרף משתנה 70

8 בעיית שימור הסדר **71**
8.1 הגדרת הבעייה 71
8.2 ניסיון ראשון 71
8.3 ניסיון שני 71
8.3.1 הרעיון הכללי 71
8.3.2 הגדרות 72
8.3.3 תיאור האלגוריתם 73
8.3.4 ניתוח סיבוכיות 73

9 זיהוי מעגלים מצטברים **76**
9.1 מיון טופולוגי של הגרף 76
9.2 אנליזה 76
9.3 חיפוש דו־כיווני 77
9.4 חיפוש חד־כיווני לגרפים צפופים 78

10 Suffix Trees **81**
10.1 Trie 81
10.2 Suffix Tree 81
10.3 בניית Suffix Tree בזמן לינארי 84
10.4 Suffix Links 85
10.5 ניתוח 86
10.6 Suffix Arrays 87

פרק 1

עצי הרחבה

בפרק זה נתאר עצי הרחבה (*Splay Trees*).

1.1 מוטיבציה

המטרה שלנו היא לתמוך באותן פעולות כמו עצי חיפוש שאנחנו מכירים, אבל בצורה יעילה ופשוטה. אנחנו רוצים להשתמש בעצים בינאריים, פשוטים, עם תכונת amortized טובה ובצורה אלגנטית.

הרעיון העיקרי שעומד מאחורי מבנה הנתונים הוא החזקת האיברים שנעשה בהם שימוש תדיר קרוב לשורש של העץ. לשם כך, נניח כי האיברים שמורים בתוך כל צומת בעץ. מאוחר יותר נוכל להרחיב את המבנה למצב שבו האיברים נמצאים בעלים.

הפעולות אותן אנחנו דורשים ממבנה הנתונים שלנו:

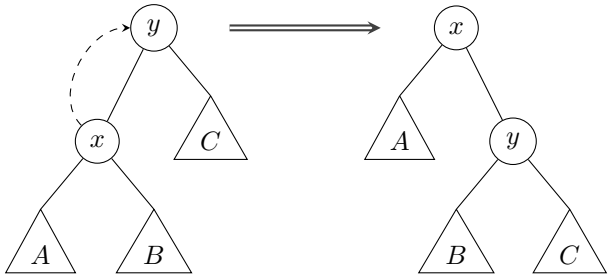
- $\texttt{Catenate}\,(T_1, T_2)$: מחבר את שני העצים הבינאריים T_1 ו־T_2 לעץ אחד.
- $\texttt{Split}\,(i, T)$: מפצלים את T לשני עצים T_1 ו־T_2, כך ש־$\forall j \in T_1.j \leq i$ ו־$\forall j \in T_2.j > i$.
- $\texttt{Insert}\,(i, T)$: מכניס את i לתוך T.
- $\texttt{Delete}\,(i, T)$: מוחק את i מ־T.

כל אחת מהפעולות צריכה להיתמך ב־$\mathcal{O}\left(\log n\right)$, כאשר n הוא מספר האיברים בעץ.

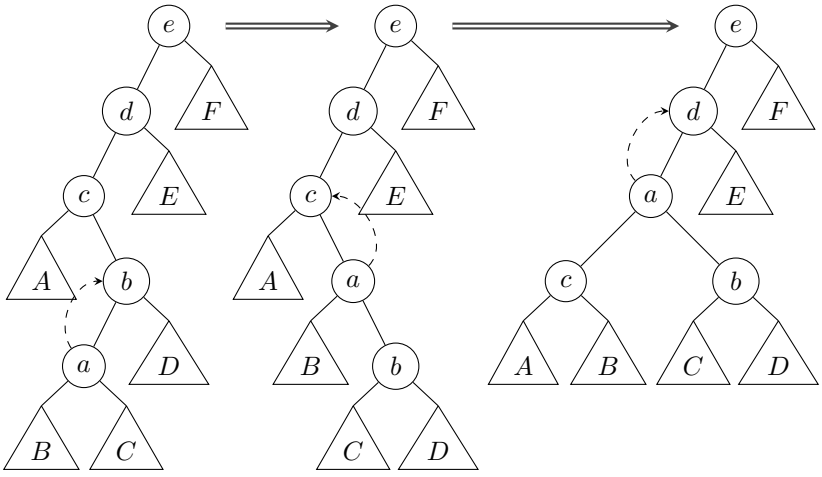
1.2 ניסיון מימוש ראשון

נעביר את האיברים שאליהם ניגשים לשורש העץ באמצעות סיבובים, מהסוג שניתן לראות באיור 1.1.

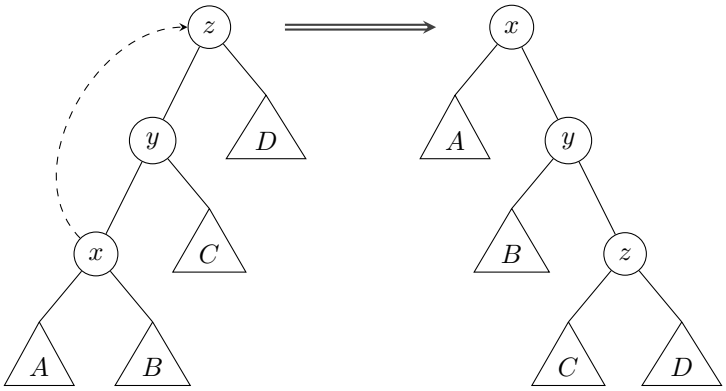
ניתן לראות דוגמה לרצף שלם של סיבובים באיור 1.2. כפי שקל לראות מהאיור, הבעיה העיקרית בשיטה זו של סיבובי עצים היא הדרישה ל־$\Omega\left(n\right)$ פעולות. באופן דומה, סדרה של n גישות דורשת $^{n^2}/_2$ פעולות.



איור 1.1: סיבוב העץ, כך ש־x יהיה בשורש



איור 1.2: רצף שלם של סיבובים בעץ, כך ש־a יהפוך להיות השורש. קל לראות שהסיבוכיות של הפעולה היא $\Omega(n)$.



איור 1.3: דוגמה ל־Zig-zig, כאשר ניגשים לצומת x.

1.3 ניסיון מימוש שני - הרחבה (*Splaying*)

במקום לבצע סיבוב על כל צומת, נבצע סיבובים בזוגות.

יש לנו שישה מקרים לביצוע Splay, כאשר הם מתחלקים לזוגות סימטריים (אין חשיבות אמיתי לימין ושמאל). שלושת המקרים (ששקולים לשישה מקרים) הם:

Zig-zig הצומת אליו ניגשים הוא הבן השמאלי, וגם אבא שלו הוא הבן השמאלי. דוגמה למצב זה ניתן לראות באיור 1.3.

Zig-zag הצומת אליו ניגשים הוא הבן הימני, ואבא שלו הוא הבן השמאלי. דוגמה למצב זה ניתן לראות באיור 1.4.

Zig האבא של הצומת אליו ניגשים הוא השורש. במצב זה מבצעים סיבוב רגיל, כפי שראינו קודם[1].

קל לראות מדוגמאות של עצים שהסיבוכיות בעזרת Splay היא הרבה יותר קטנה מאשר סיבובים רגילים. ניתן לראות דוגמה באיורים 1.5 ו־1.6.

כמו כן, כאשר מבצעים רק Zig-zag, או רק Zig-zig, העצים נהיים מאוזנים יחסית. ניתן לראות זאת באיור 1.7 ו־1.8 בהתאמה.

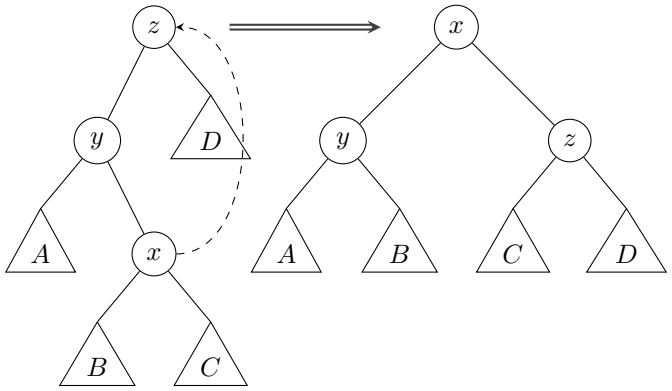
1.3.1 אנליזה

נניח שלכל איבר i יש משקל חיובי $w(i)$, שהוא שרירותי אבל קבוע.

הגדרה 1 (גודל של צומת): הגודל $s(x)$ של הצומת x בעץ הוא סכום המשקלים של כלל האיברים בתת־העץ של x.

הגדרה 2 (דרגה של צומת): הדרגה של הצומת x, המסומנת ב־$r(x)$ היא $\log_2(s(x))$.

[1] ניתן לראות דוגמה גרפית לכך באיור 1.1.



איור 1.4: דוגמה ל-Zig-zag, כאשר ניגשים לצומת x.

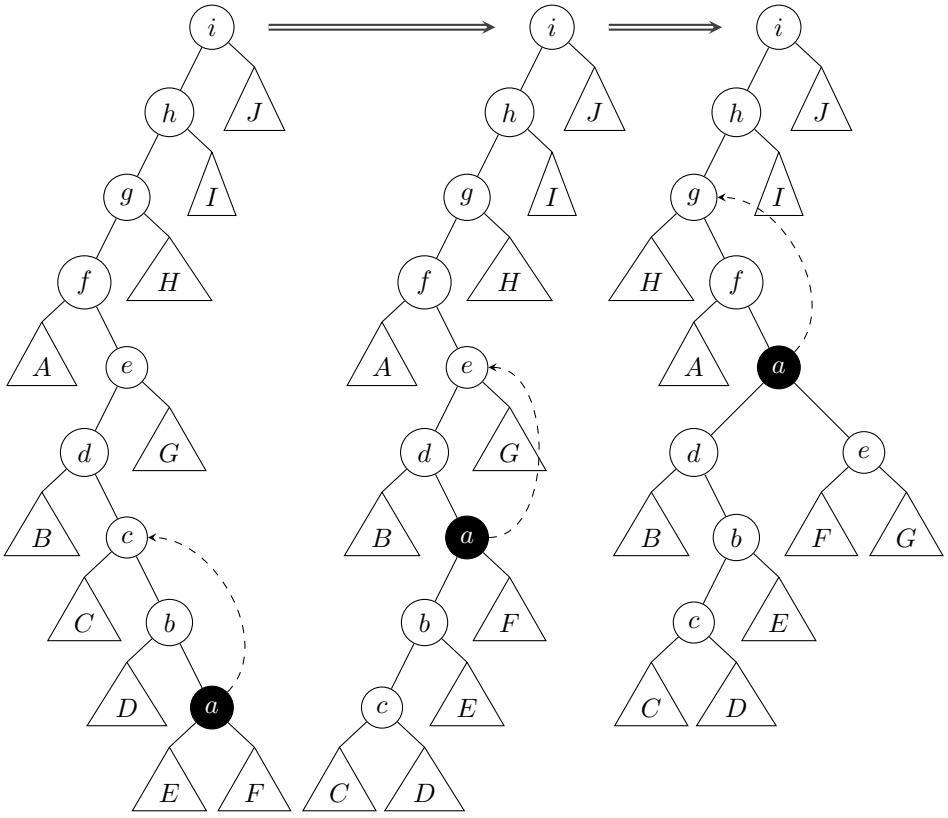
נמדוד את הזמן שלוקח Splay לפי מספר הסיבובים שעושים.

למה 3 (למת הגישה): זמן ה-*Amortized* לביצוע *Splay* לצומת x בעץ שהשורש שלו הוא t הוא לכל היותר $3 \cdot (r(t) - r(x)) + 1 = 3 \cdot \log_2\left(\frac{s(t)}{s(x)}\right) + 1$. פונקציית הפוטנציאל לחישוב ה-*Amortized* היא סכום הדרגות בעץ: $\sum r(x) = \sum \log(s(x))$.

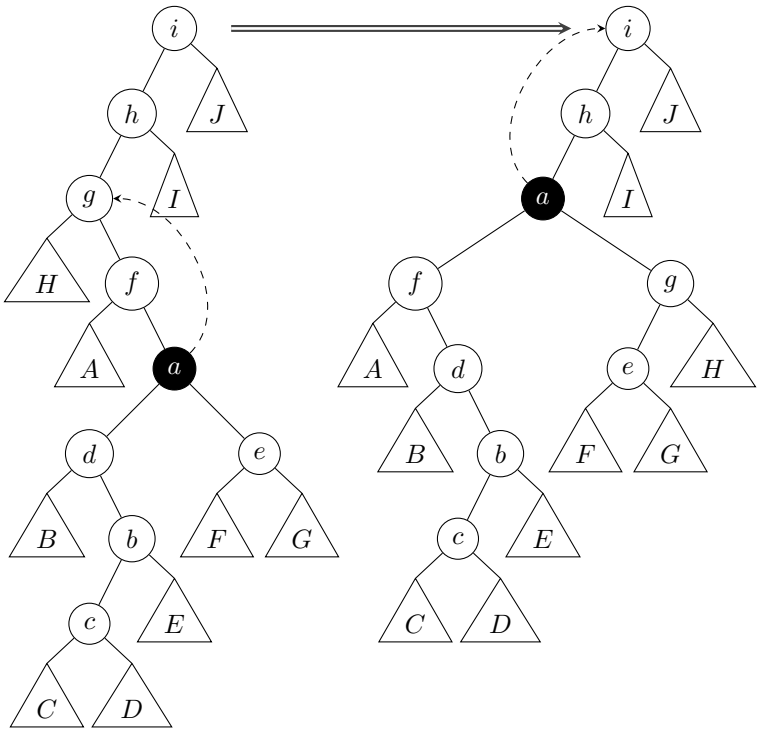
הוכחה (למת הגישה): ניתן למצוא אינטואיציה להוכחה באיורים 1.9 ו-1.10.

נסתכל על צעד של Splay. יהיו s ו-r הגודל והדרגה של כל העץ לפני ה-Splay, בהתאמה, ו-s' ו-r' הגודל והדרגה של כל העץ אחרי ה-Splay, בהתאמה. נראה שזמן ה-Amortized של צעד מסוג Zig הוא לכל היותר $3 \cdot (r'(x) - r(x)) + 1 = 3 \cdot \Delta\Phi(x) + 1$, וזמן ה-Amortized של צעדים מסוג Zig-Zig ו-Zig-Zag הוא לכל היותר $3 \cdot (r'(x) - r(x)) = 3 \cdot \Delta\Phi(x)$.

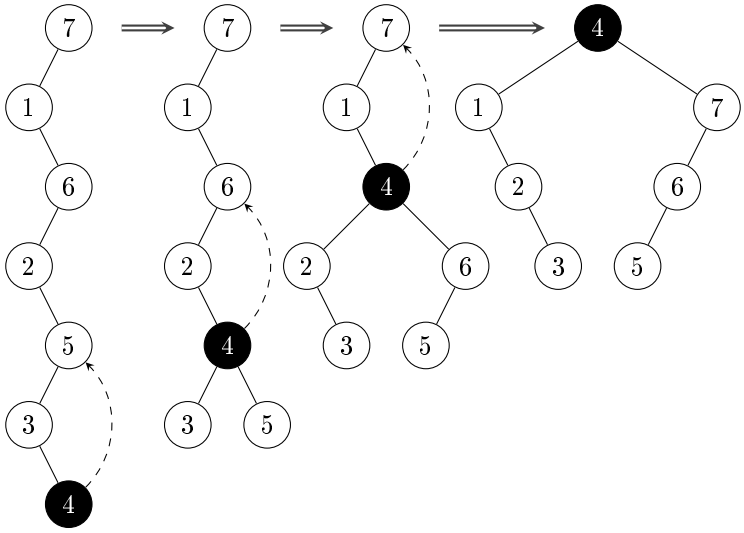
לאחר שנוכיח את זה, מסכימת העלויות של כל צעדי ה-Splay שנעשה, נקבל את הלמה. נניח כי מבצעים Splay לצומת x, והשורש הוא הצומת t. אזי אנחנו מקבלים טור טלסקופי, ונשארים עם הגורמים של x ו-t בלבד.



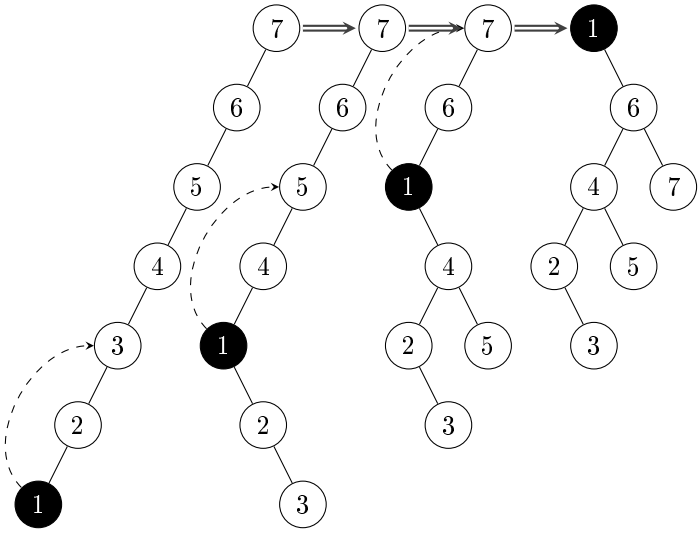
איור 1.5: דוגמה לביצוע Splay שמביא את a לשורש. קל לראות שהסיבוכיות ירדה ל־$\mathcal{O}(\log n)$



איור 1.6: המשך הדוגמה לביצוע Splay שמביא את a לשורש

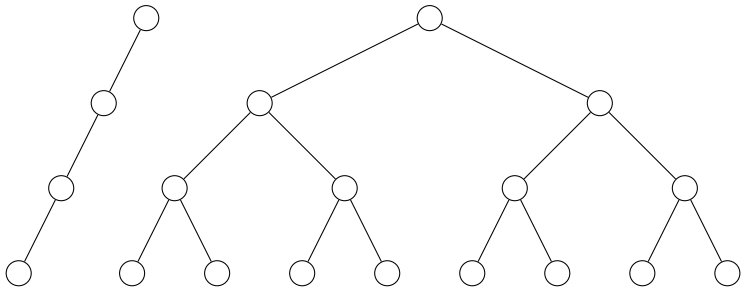


איור 1.7: ביצוע פעולות Zig-zag.



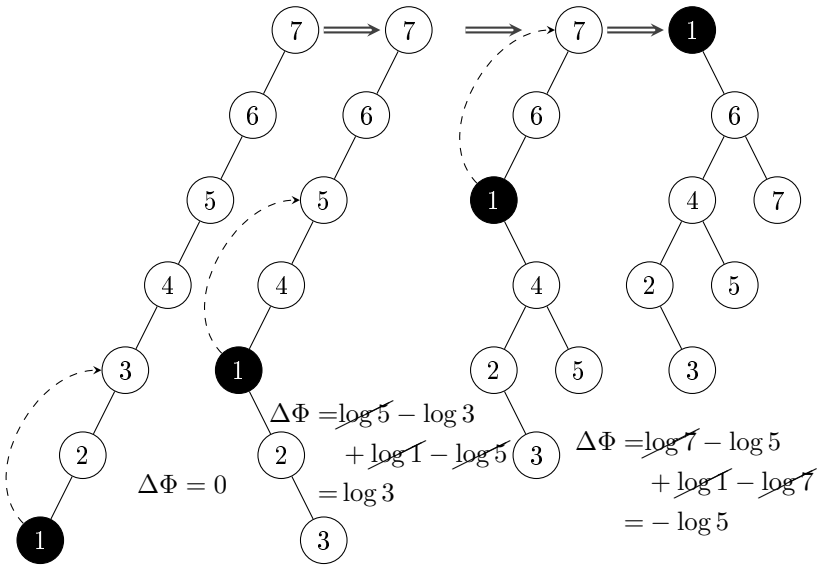
איור 1.8: ביצוע פעולות Zig-zig.

$\Phi = \sum_{i=1}^{n} \log i = \mathcal{O}(n \cdot \log n)$



$\Phi = \frac{n+1}{2} \cdot \log 1 + \frac{n+1}{4} \cdot \log 3 + \cdots + \log n \leq (n+1) \cdot \sum_i \frac{i}{2^i} = \mathcal{O}(n)$

איור 1.9: ניתן לראות בבירור את ההבדלים בין עצים מאוזנים לעצים שאינם מאוזנים בפוטנציאל. הפוטנציאל של עץ מאוזן הוא $\mathcal{O}(n)$, בעוד הפוטנציאל של עץ לא מאוזן הוא $\mathcal{O}(n \cdot \log n)$.



איור 1.10: דוגמה לביצוע Splay-ים בעץ, וההשפעה שלהם על הפוטנציאל. ניתן לראות בקלות שכל Splay מאזן את העץ קצת יותר, ויחד עם זה מוריד את הפוטנציאל.

עובדה 4:

$$\begin{aligned}
0 \leq (a-b)^2 &= a^2 - 2\cdot a\cdot b + b^2 \\
&\Downarrow \\
2\cdot a\cdot b &\leq a^2 + b^2 \\
&\Downarrow \\
4\cdot a\cdot b &\leq a^2 + 2\cdot a\cdot b + b^2 = (a+b)^2 \\
&\Downarrow \\
\log(4\cdot a\cdot b) &\leq \log(a+b)^2 \\
\underbrace{\log 4}_{2} + \log a + \log b &\leq 2\cdot\log(a+b) \\
\log a + \log b &\leq 2\cdot\log(a+b) - 2
\end{aligned}$$

נסתכל על כל אחד מהמקרים האפשריים:

1. Zig:

העלות האמיתית של צעד Zig היא 1.

נסתכל על השינוי הפוטנציאל (לפי הסימונים באיור 1.1):

$$\begin{aligned}
\Delta\Phi(T) &= \cancel{\Phi'(x)} + \Phi'(y) - \Phi(x) - \cancel{\Phi(y)} \\
&= \Phi'(y) - \Phi(x) \leq \Phi'(x) - \Phi(x) \\
&= \Delta\Phi(x) \leq 3\cdot\Delta\Phi(x)
\end{aligned}$$

לכן, עלות ה-Amortized של צעד Zig היא לכל היותר $3\Delta\Phi(x)+1$.

2. Zig-Zag:

העלות האמיתית של צעד Zig-Zag היא 2.

נסתכל על השינוי הפוטנציאל (לפי הסימונים באיור 1.7):

$$\begin{aligned}
\Delta\Phi(T) &= \cancel{\Phi'(x)} + \Phi'(y) + \Phi'(z) - \Phi(x) - \Phi(y) - \cancel{\Phi(z)} \\
&= \Phi'(y) + \Phi'(z) - \Phi(x) - \Phi(y) \\
&\leq 2\cdot\Phi'(x) - 2 - 2\cdot\Phi(x) \qquad (1.1) \\
&\leq 2\cdot\Delta\Phi(x) - 2
\end{aligned}$$

כאשר המעבר לשורה 1.1 בוצע על פי עובדה 4, בצירוף עם הדברים הבאים:

$$\begin{aligned}
s'(y) + s'(z) &= s'(x) \\
s(y) \geq s(x) &\Rightarrow -\Phi(y) \leq -\Phi(x)
\end{aligned}$$

לכן, עלות ה-Amortized של צעד Zig-Zag היא $\cancel{2} + 2\cdot\Delta\Phi(x) - \cancel{2} = 2\cdot\Delta\Phi(x) \leq 3\cdot\Delta\Phi(x)$.

3. Zig-Zig:

העלות האמיתית של צעד Zig-Zig היא 2.

נסתכל על השינוי בפוטנציאל (לפי הסימונים באיור 1.3):

$$\begin{aligned}\Delta\Phi(T) &= \cancel{\Phi'(x)} + \Phi'(y) + \Phi'(z) - \Phi(x) - \Phi(y) - \cancel{\Phi(z)} \\ &= \Phi'(y) + \Phi'(z) - \Phi(x) - \Phi(y) \\ &= \Phi'(y) + \Phi'(z) + \Phi(x) - 2\cdot\Phi(x) - \Phi(y) \\ &\leq \Phi'(x) + 2\cdot\Phi'(x) - 2 - 3\cdot\Phi(x) \qquad (1.2) \\ &= 3\cdot\Delta\Phi(x) - 2\end{aligned}$$

כאשר המעבר לשורה1.2 בוצע לפי עובדה 4, בצירוף עם הדברים הבאים:

$$\begin{aligned}\Phi'(y) &\leq \Phi'(x) \\ s'(z) + s(x) &= \underbrace{w(z) + s(C) + s(D)}_{s'(z)} + \underbrace{w(x) + s(A) + s(B)}_{s(x)} \\ &= s'(x) - w(y) \leq s'(x) \\ \Phi(y) \geq \Phi(x) &\Rightarrow -\Phi(y) \leq -\Phi(x)\end{aligned}$$

לכן, עלות ה-Amortized של צעד Zig-Zig היא $3\cdot\Delta\Phi(x) - 2 \leq 3\cdot\Delta\Phi(x)$.

הוכחנו כי כל בכל מקרה, זמן ה-Amortized של פעולת Splay לא עולה על $3\cdot\Delta\Phi(x)+1$.

■

1.3.2 שימושים

מלמת הגישה, ניתן להסיק משפטים נוספים:

משפט 5 (משפט האיזון): גישה ל-m איברים בעץ *Splay* עם n צמתים לוקחת זמן של $\mathcal{O}((m+n)\cdot\log n)$.

הוכחה: ניתן לכל צומת את המשקל 1. לכן, המשקל הכולל יהיה $W = n$. כמו כן, מלמת הגישה, ביצוע של Splay אורך $3\cdot\log n + 1$ בזמן Amortized.
לכן, השינוי הכולל בפוטנציאל הוא לכל היותר $n\cdot\log n$. ■

משפט 6 (Static Optimality Theorem): לכל איבר i, נסמן ב-$q(i)$ את מספר הגישות ל-i. אם ניגשים לכל איבר לפחות פעם אחת, אזי זמן הגישה הכולל הוא $\mathcal{O}\left(m + \sum_{i=1}^{n} q(i)\cdot\log\frac{m}{q(i)}\right)$.

הערה ?: זהו זמן גישה אופטימלי, עד כדי קבוע.

הוכחה: ניתן לאיבר ה-i את המשקל $\frac{q(i)}{m}$. אזי $W = \sum_{i=1}^{n} w(i) = 1$.
מלמת הגישה, זמן ה-Amortized לביצוע Splay ל-i הוא $1 + \frac{m}{q(i)}\cdot\log 3$. לכן, השינוי הכולל בפוטנציאל על הסדרה הוא:

$$\sum_{i=1}^{n} \log W - \log\frac{q(i)}{m}$$

■

משפט 8 (Static Finger Theorem): נניח שכל האיברים מסומנים מ־1 עד i בסדר סימטרי. נניח כי $i_1, \ldots, i_m$ היא סדרת הגישות לאיברים (כאשר i_j הוא האיבר שאליו ניגשים בגישה ה־j).

יהי f איבר קבוע. אזי זמן הגישה הכולל הוא:

$$\mathcal{O}\left(n \cdot \log n + m + \sum_{i=1}^{m} \log\left(|i_j - f| + 1\right)\right)$$

מסקנה 9: עצי *Splay* מאפשרים גישה בסביבה של כל איבר מקובע בצורה טובה כמו *Finger search trees*.

משפט 10 (Working Set Theorem): יהי $t(j)$ (עבור $j \in \{1, \ldots, m\}$) מספר האיברים השונים שאליהם נרשמה גישה מאז הגישה אחרונה ל־j, או מאז תחילת סדרת הגישות.

אזי זמן הגישה הכולל הוא:

$$\mathcal{O}\left(n \cdot \log n + m + \sum_{j=1}^{m} \log\left(t(j) + 1\right)\right)$$

הוכחה: ניתן את המשקלים $1, 1/4, 1/9, \ldots, 1/k^2$ לפי סדר הגישות. אחרי גישה, נגיד לאיבר במשקל k לשנות את המשקל שלו, כך שהמשקל שלו יהיה 1, וכל איבר ממשקל $\frac{1}{d^2}$ יהיה ממשקל $\frac{1}{(d+1)^2}$ לכל $d < k$.

המשקלים ששונו אחרי ה־Splay רק הקטינו את הפוטנציאל. הפוטנציאל הוא אי־חיובי, ולפחות $-2 \cdot n \cdot \log n$. ■

1.4 שימושים

1.4.1 דחיסה באמצעות עצי Splay

נניח שאנחנו רוצים לדחוס מחרוזות מעל האלפבית Σ. אלגוריתם 1.1 מתאר את מנגנון הדחיסה והפריסה של המחרוזות.

נבחן את טיב האלגוריתם: נניח כי m הוא מספר התווים במחרוזת המקורית. האורך של המחרוזת שנייצר הוא $m + c$, כאשר c הוא העלות של ביצוע פעולות ה־Splay. ממשפט 6:

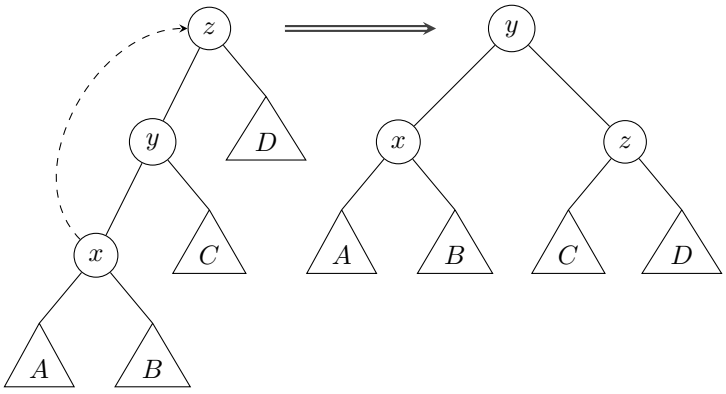
$$m + \mathcal{O}\left(m + \sum q(i) \cdot \log \frac{m}{q(i)}\right) = \mathcal{O}\left(m + \sum q(i) \cdot \log \frac{m}{q(i)}\right)$$

עובדה 11: האנטרופיה של הסדרה $\sum q(i) \cdot \log \frac{m}{q(i)}$ היא חסם תחתון.

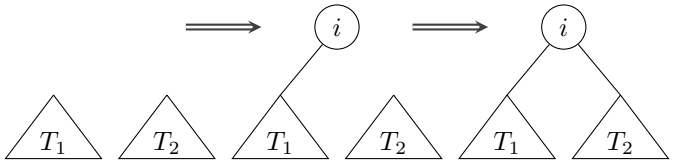
בפרט, קידוד Huffman של סדרה הוא $\sum q(i) \cdot \log \frac{m}{q(i)}$, אבל כדי לבנות אותו צריך לדעת את התדירויות של כל תו מראש.

אלגוריתם 1.1 דחיסה באמצעות עצי Splay

```
function Prepare(Alphabet Σ)
    return Binary tree containing the items of Σ at its leaves
end function
function EncodeSymbol(Symbol x ∈ Σ, State tree T)
    v ← Root(T), e ← Empty string
    while not IsLeaf(v) do
        if PathTo(x, v) = Left(v) then e ← e‖0
        else e ← e‖1
        end if
        v ← PathTo(x,v)
    end while                    ▷ Now v if the leaf of x. Splay at the parent of v
    v ← Parent(v)
    Splay(v)
    return e
end function
function DecodeSymbol(Encoded string e, State tree T, Offset i)
    v ← Root(T)
    while not IsLeaf(v) do
        if e[i] = 1 then v ← Left(v)
        else v ← Right(v)
        end if
        i ← i + 1
    end while
        ▷ Now v is the leaf of the decoded symbol x. Splay at the parent of v
    x ← ExtractSymbol(v)
    v ← Parent(v)
    Splay(v)
    return x, i
end function
function EncodeString(Alphabet Σ, String s ∈ Σ^n)
    T ← Prepare(Σ), e ← Empty string
    for all c ∈ s do e ← e‖EncodeSymbol(c, T)
    end for
    return e
end function
function DecodeString(Alphabet Σ, String e)
    T ← Prepare(Σ), s ← Empty String, i ← 1
    while i < Length(e) do
        c, i ← DecodeSymbol(e, T, i)
        s ← s‖c
    end while
    return s
end function
```



איור 1.11: דוגמה ל־Semi Zig-zig, כאשר ניגשים לצומת x.



איור 1.12: ביצוע פעולת Catenate לעצי Splay

1.4.2 דחיסה באמצעות עצי Semi-Splay

D. Jones הראה בשנת 1988 שדחיסה באמצעות עצי Splay יכולה להתחרות בקידוד Huffman, באמצעות גרסה אחרת של עצי Splay, שנקראת עצי Semi-Splay. בעצים כאלו, משנים את פעולת ה־Zig-Zig, כך שתהיה כמו באיור 1.11.

1.5 פעולות על עצי Splay

1.5.1 Catenate

פעולת $\texttt{Catenate}(T_1, T_2)$ מחברת את העצים T_1 ו־T_2 לעץ חיפוש בינארי תקין.
את הביצוע של הפעולה ניתן לראות באלגוריתם 2.1, ואת התיאור הגרפי שלה ניתן לראות באיור 1.12.

אלגוריתם 2.1 פעולת `Catenate` על עץ Splay

```
  function CATENATE(T_1, T_2)
Require: T_1 and T_2 are splay trees, and ∀i ∈ T_1, j ∈ T_2.i < j
    i ← MAXITEM(T_1)                                  ▷ The rightmost leaf
    SPLAY(i)
    RIGHT(i) ← ROOT(T_2)
  end function
```

אלגוריתם 3.1 ביצוע פעולת $\texttt{Split}\,(i,T)$ כאשר $i \in T$

```
  function SPLIT(Node i, Splay tree T)
Require: i ∈ T
    SPLAY(i)
    T_2 ← RIGHT(i)
    RIGHT(i) ← nil
    T_1 ← i
    return T_1, T_2
  end function
```

זמן Amortized לפעולת `Catenate`:

$$\begin{aligned}\text{Amortized Time} &= 3\cdot\left(\log\frac{s\left(T_1\right)}{s\left(i\right)}+1+\log\frac{s\left(T_1\right)+s\left(T_2\right)}{s\left(T_1\right)}\right)\\ &\leq 3\cdot\log\frac{W}{w\left(i\right)}+\mathcal{O}\left(1\right)\end{aligned}$$

`Split` 1.5.2

ניתן לחלק את הפעולה לשני מקרים: $i \in T$ ו־$i \notin T$. במימוש בסוף, לא תהיה לכך השפעה.

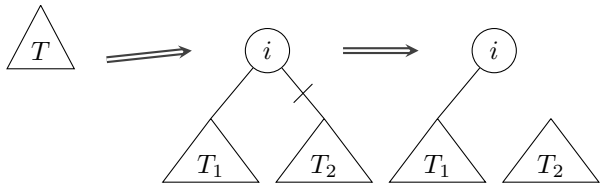
$\texttt{Split}\,(i,T)$ **כאשר** $i \in T$ 1.5.2.1

אלגוריתם 3.1 ואיור 1.13 מתארים ביצוע של פעולת $\texttt{Split}\,(i,T)$ כאשר $i \in T$.
זמן Amortized:

$$\text{Amortized Time} = 3\cdot\log\frac{W}{w\left(i\right)}+\mathcal{O}\left(1\right)$$

$\texttt{Split}\,(i,T)$ **כאשר** $i \notin T$ 1.5.2.2

במידה ו־$i \notin T$, נחפש את i בכל זאת בעץ, ונגיע לאיבר שהוא הכי גדול שקטן מ־i (נקרא לו i^-), או הכי קטן שגדול מ־i (נקרא לו i^+). עליו נבצע את ה־Splay.



איור 1.13: פעולת $\texttt{Split}\,(i,T)$ כאשר $i \in T$

אלגוריתם 4.1 ביצוע פעולת $\texttt{Split}\,(i,T)$ כאשר $i \notin T$

```
function SPLIT(Node i, Splay tree T)
    j ← Nearest element of i in T
    SPLAY(j)
    if j = i⁻ or j = i then
        T2 ← RIGHT(j)
        RIGHT(j) ← nil
        T1 ← j
    else
        T1 ← LEFT(j)
        LEFT(j) ← nil
        T2 ← j
    end if
    return T1, T2
end function
```

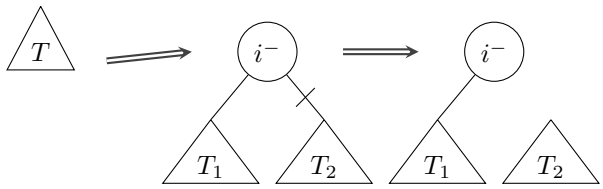
אלגוריתם 4.1 ואיור 1.14 מתארים את ביצוע פעולת $\texttt{Split}\,(i,T)$ כאשר $i \notin T$.
זמן Amortized:

$$\text{Amortized Time} = 3 \cdot \log \frac{W}{\min\left\{w\left(i^{-}\right), w\left(i^{+}\right)\right\}} + \mathcal{O}\left(1\right)$$

1.5.3 Insert

אלגוריתם 5.1 מתאר את ביצוע פעולת $\texttt{Insert}\,(i,T)$. איור 1.15 מתאר את העץ המוחזר מפעולת $\texttt{Insert}\,(i,T)$.
זמן Amortized:

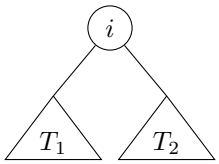
$$\text{Amortized Time} = 3 \cdot \log \frac{W}{\min\left\{w\left(i^{-}\right), w\left(i^{+}\right)\right\}} + \log \frac{W}{w\left(i\right)} + \mathcal{O}\left(1\right)$$



איור 1.14: פעולת $\texttt{Split}\,(i,T)$ כאשר $i \notin T$ (כאשר מבצעים Splay על i^-)

אלגוריתם 5.1 ביצוע $\texttt{Insert}\,(i,T)$ בעץ Splay

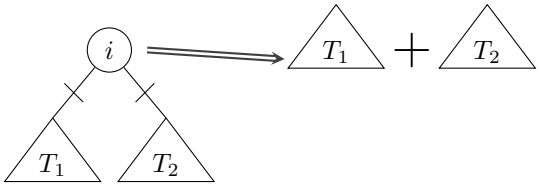
```
function INSERT(Node i, Splay tree T)
    T_1, T_2 ← SPLIT(i, T)
    LEFT(i) ← T_1
    RIGHT(i) ← T_2
    return i
end function
```



איור 1.15: העץ המוחזר מפעולת $\texttt{Insert}\,(i,T)$

אלגוריתם 6.1 ביצוע פעולת $\texttt{Delete}\,(i,T)$ בעץ Splay

```
  function DELETE(Node i, Splay tree T)
Require: i ∈ T
    SPLAY(i)
    T_1 ← LEFT(i)
    T_2 ← RIGHT(i)
    LEFT(i) ← nil
    RIGHT(i) ← nil
    return T_1, T_2
  end function
```



איור 1.16: ביצוע פעולת $\texttt{Delete}\,(i,T)$ בעץ Splay

Delete 1.5.4

אלגוריתם 6.1 ואיור 1.16 מתארים את ביצוע פעולת $\texttt{Delete}\,(i,T)$.
זמן Amortized:

$$\text{Amortized Time} = 3 \cdot \log \frac{W}{w\left(i\right)} + 3 \cdot \log \frac{W - w\left(i\right)}{w\left(i^{-}\right)} + \mathcal{O}\left(1\right)$$

1.6 שאלות פתוחות

Dynamic Optimality Conjecture 1.6.1

נבחן כל סדרת חיפושים מוצלחים בעץ בינארי עם n צמתים. יהי $\mathcal{A}$ כל אלגוריתם שמבצע חיפושים באמצעות חציית המסלול מהשורש לצומת שמכיל את האיבר בעלות של 1 ועוד העומק של הצומת שמכיל את האיבר, ושבין שני חיפושים מבצע סיבובים במקום כלשהו בעץ, בעלות של 1 לכל סיבוב.

אזי הזמן הכולל לביצוע כל החיפושים האלה לא עולה על $\mathcal{O}\left(n\right)$ ועוד עלות הביצוע של $\mathcal{A}$ כפול קבוע כלשהו.

1.6.2 Dynamic Finger Conjecture[2]

הזמן הכולל לביצוע m חיפושים מוצלחים על עץ Splay כלשהו עם n צמתים הוא:

$$\mathcal{O}\left(m+n+\sum_{j=1}^{m}\left(\log|i_{j+1}-i_j|+1\right)\right)$$

כאשר הגישה ה־j־ית היא לאיבר i_j.
ההוכחה מורכבת מאוד, והיא הוצגה בזמן האחרון.

[2] היום זה כבר משפט.

פרק 2

עצים דינאמיים

מבנה הנתונים נועד לתחזק יער של עצים מכוונים‌ץ לכל עץ ביער יש שורש. לכל קשת (v,w) יש משקל (עלות, cost) $c(v,w)$.

2.1 פעולות נתמכות

מבנה הנתונים צריך לתמוך בפעולות הבאות:

`maketree` (v) יוצר עץ עם צומת בודד v.

`link` $(v,w,c(v,w,))$ v הוא שורש של עץ ו־w הוא צומת בעץ אחר. הפעולה תולה את v על w, ונותנים לקשת (v,w) את המשקל $c(v,w)$. ניתן לראות דוגמה באיור 2.1.

`cut` (v) מנתק את הקשת מ־v לאבא של v. ניתן לראות דוגמה באיור 2.1.

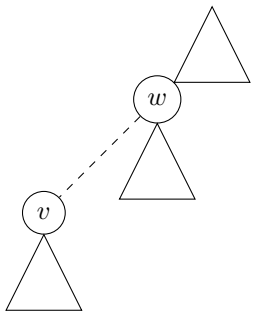
$w=$ `findroot` (v) מחזיר את השורש w של העץ ש־v הוא צומת שלו.

$(w,c)=$ `mincost` (v) (יכול להיות גם `maxcost`) מחזיר את הצומת w שהקשת שיוצאת ממנו $(w,p(w))$ היא בעלת עלות c, וזוהי העלות הקטנה ביותר של קשת על המסלול מ־v לשורש. אם יש כמה קשתות כאלה, נחזיר את הצומת הקרוב ל־v.

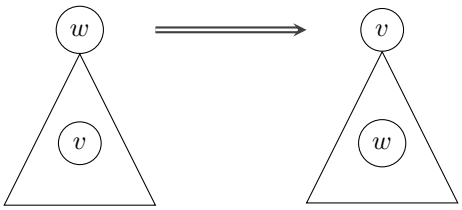
`addcost` (v,c) מוסיף את הקבוע c לעלות של כל אחת מהקשתות שעל המסלול מ־v לשורש.

`evert` (v) הופך את v להיות השורש, על ידי שינוי כיווני הקשתות. בצורה פורמלית: משנים את שורש העץ שבו נמצא v להיות v, על ידי היפוך כיוון הקשתות על המסלול מ־v לשורש. ניתן לראות דוגמה באיור 2.2.

קיים מימוש של עצים דינאמיים, המאפשר ביצוע של כל אחת מהפעולות בזמן Amortized של $\mathcal{O}(\log n)$, כאשר n הוא מספר הצמתים.



איור 2.1: דוגמה לפעולת $\texttt{link}\left(v,w,c\left(v,w\right)\right)$ או $\texttt{cut}\left(v\right)$



איור 2.2: דוגמה לפעולת $\texttt{evert}\left(v\right)$

2.2 שימושים

2.2.1 עץ פורש מינימלי

תחזוק של עץ פורש מינימלי לגרף שכל הזמן מוסיפים לו קשתות (Online). כלומר, מוסיפים קשתות אחת אחרי השניה לגרף (לא מכוון) עם משקל. רוצים בכל רגע נתון יער פורש מינימלי של הגרף.

תזכורת 12 (תזכורת): קשת (v,w) נמצאת בעץ פורש מינימלי אם ורק אם קיים חתך כך ש־(v,w) היא המינימלית שחוצה אותו.

תזכורת 13 (תזכורת): קשת אינה בעץ פורש מינימלי אם ורק אם היא מקבימלית על איזשהו מעגל.

נשתמש בעץ דינאמי: נתחזק את היער הפורש המינימלי בעזרתו. ה־$c(v,w)$ בעץ הדינאמי הוא המשקל של הקשת, והשורשים יהיו שרירותיים.

איך נעשה את זה? אם הקשת שמכניסים עכשיו חוצה רכיבי קשירות (כלומר מחברת שני עצים), נגלה זאת בעזרת `findroot`, ואז נחבר את העצים (בעזרת `evert` ו־`link`). אם v ו־w הם באותו הרכיב, נשווה את העלות של הקשת החדשה לעלות של הקשת הכי יקרה במסלול בין v ו־w (בעזרת `evert` ו־`maxcost`). אם צריך להוסיף את הקשת, נמחק את הקשת היקרה ביותר (בעזרת `cut`), ובכל מקרה נוסיף את הקשת החדשה (בעזרת `link`, כי v הוא כבר השורש).

פתרנו את בעיית התחזור של יער פורש מינימלי של גרף דינאמי המשתנה על ידי הוספת קשתות בלבד, על ידי ביצוע של מספר קבוע של פעולות על עצים דינאמיים לכל הכנסה של קשת. אם נדע לממש עצים דינאמיים בעלות $\mathcal{O}(\log n)$ לפעולה, סך הכל של הכנסת m קשתות תהיה $\mathcal{O}(m \cdot \log n)$.

2.2.2 עץ פורש מינימלי עם שני צבעים

נניח שיש גרף שבו הקשתות צבועות בכחול ושחור, בנוסף על המשקל שלהן. רוצים עץ פורש מינימלי עם מספר נתון של קשתות כחולות. למשל, אם רוצים שלצומת תהיה דרגה קטנה בעץ הפורש המינימאלי, צובעים את הקשתות הסמוכות לו בכחול.

נסתכל על המקרה הפרטי: רוצים כמה שיותר קשתות כחולות. נקטין את המשקל של הקשתות הכחולות, כך שיהיה יותר קטן מהמשקל של הקתות השחורות (למשל $-\infty$), ואז נחפש עץ פורש מינימאלי. זהו מקרה פרטי למקרה שבו מוסיפים λ למשקל של כל קשת כחולה, כאשר במקרה הזה, $\lambda = -\infty$. אם כל פעם נגדיל את λ, אפשר למצוא כל מספר של קשתות כחולות בעץ הפורש המינימאלי.

הגדרה 14 (ערך קריטי): λ הוא ערך קריטי אם מספר הקשתות הכחולות בעץ הפורש המינימאלי המקבל מהוספת המשקל $\lambda-\varepsilon$ לכל הקשתות הכחולות בגרף שונה ממספר הקשתות הכחולות בעץ הפורש המינימאלי המתקבל מהוספת המשקל λ לכל הקשתות הכחולות בגרף.

בעזרת עצים דינאמיים, נאתחל יער פורש מינימאלי עם קשתות כחולות בלבד. נסדר את הקשתות השחורות לפי משקל עולה. כאשר מטפלים בקשת שחורה e, מבצעים את אחד מהמקרים הבאים:

1. הקשת השחורה סוגרת מעגל ביער (ניתן לזיהוי בעזרת `findroot`).

   במקרה זה, מוציאים את הקשת הכחולה המקסימאלית e' מהמעגל (מוצאים בעזרת `evert` ו־`maxcost`, ומוציאים בעזרת `cut`), ומחליפים אותה ב־e (בעזרת `link`). במקרה זה, $c(e)-c(e')$ הוא ערך קריטי.

2. הקשת השחורה סוגרת מעגל עם קשתות שחורות בלבד.

   במקרה זה, ממשיכים הלאה (הקשת השחורה לעולם לא תהיה בעץ פורש מינימאלי, ללא תלות במספר הקשתות הכחולות והשחורות).

3. הקשת e מחברת שני עצים שונים.

   נצרף אותה ליער ונמשיך. זו קשת שחורה שמופיעה בכל עץ פורש מינימאלי לכל מספר רצוי של קשתות כחולות.

על ידי מיון של סדרת הערכים הקריטיים נוכל לגלות את העץ הפורש המינימאלי לכל מספר של קשתות כחולות.

2.2.3 ID Range Reporting

נתונים אינטרוולים חד־מימדיים, ולכל אחד יש עדיפות (priority). רוצים מבנה נתונים שמכיל קטעים עם עדיפות, ותומך בפעולות `insert`, `delete` ומציאת קטע עם עדיפות מינימלית שמכיל נקודה כלשהי שניתנת בשאילתה.

המוטיבציה לבעיה זו היא טיפול בפקטות בראוטר, כדי לדעת איזה פקטות לחסום. השאילתות צריכות להיות מאוד מהירות. לרוב, החסימות בראוטרים הן לפי ה־Prefix של כתובת ה־IP. לכן, נוסיף את ההנחה שכל זוג קטעים I_1 ו־I_2 מקיים $I_1 \subseteq I_2$ (ללא הגבלת הכלליות) או $I_1 \cap I_2 = \emptyset$.

באופן טבעי, מערכת שכזו מגדירה עץ: הכלה היא יחס של בן בעץ. האבא של v הוא הקטן ביותר שמכיל את v.

נחשוב על השאילתה: נסתכל על העלה s שמייצג את הקטע הכי קטן שמכיל את הנקודה. הקשתות יסמלו את עדיפויות הקטע, וכך בעזרת `mincost` נמצא את הקטע בעל העדיפות הנמוכה ביותר. אם נשים את זה בעץ בינארי, העלות תהיה $\mathcal{O}(\log n)$.

יש שתי בעיות:

1. זיהוי נקודת ההתחלה (s).

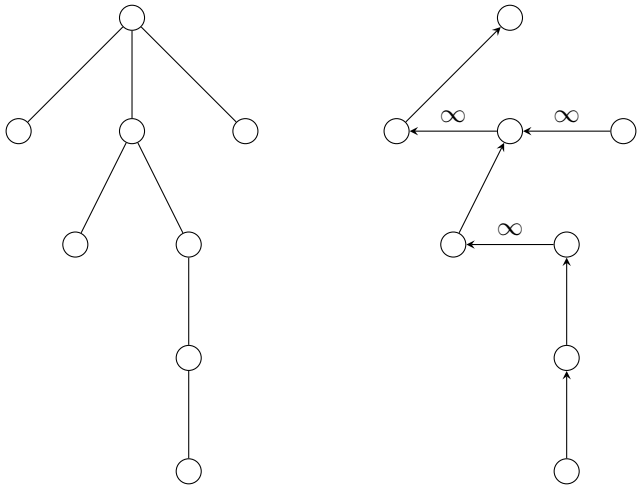
2. איך נתמוך ב־`insert` ו־`delete`?

   נוכל לעשות את זה עם הרבה `link`־ים ו־`cut`־ים, אבל זה מאוד יקר.

כדי לפתור את הבעיה, נעשה בינאריזציה (*Binarization*) לעץ, כלומר נהפוך אותו לעץ בינארי: הבן השמאלי של v יהיה הבן השמאלי שלו, וכל ילד אחר של v יהפוך להיות הבן הימני של האח השמאלי שלו. עכשיו, בחיפוש שלנו, נראה גם קטעים שלא שייכים לנקודה. לכן, כל בן ימני יתן משקל של ∞, וככה לא נקבל קטעים לא נכונים. ניתן למצוא דוגמה לבינאריזציה באיור 2.3.

השאילתה עדיין משתמשת ב־`mincost` ב־$\mathcal{O}(\log n)$ (ללא שינוי), אבל פעולות העדכון קלות יותר עכשיו. כל עדכון ידרוש מספר קבוע של `link`־ים ו־`cut`־ים.

אנחנו עדיין לא יודעים איך מוצאים את הנקודה שבה מתחילים את השאילה, ואיך מזהים איפה מכניסים את הקטע. נשתמש בעץ חיפוש מאוזן (כלשהו) על ה־endpoint של הקטעים. הסיבוכיות תישאר $\mathcal{O}(\log n)$. ייתכן שה־endpoint מייצג קטע שהנקודה לא מוכלת בו. צריך להיות מודעים לאפשרות הזו, ובמצב הזה מסתכלים על האבא של הצומת שאליו הגענו.



איור 2.3: דוגמה לבינאריזציה של עץ. קשתות שאינן בעץ המקורי, משקלן ∞.

2.3 מימוש

2.3.1 מקרה פרטי – שרוך

נניח תחילה שכל העצים הם שרוכים (או מסלולים). הייצוג של השרוף יהיה באמצעות עצי Splay, כך שהסדר בעץ מתאים לסדר בשרוך.

כדי לבצע $w = \texttt{findroot}(v)$, נעשה Splay כך ש־v יהיה השורש, ונלך ימינה כל הזמן עד שמגיעים ל־w. נחזיר את w, ונבצע עליו Splay[1].

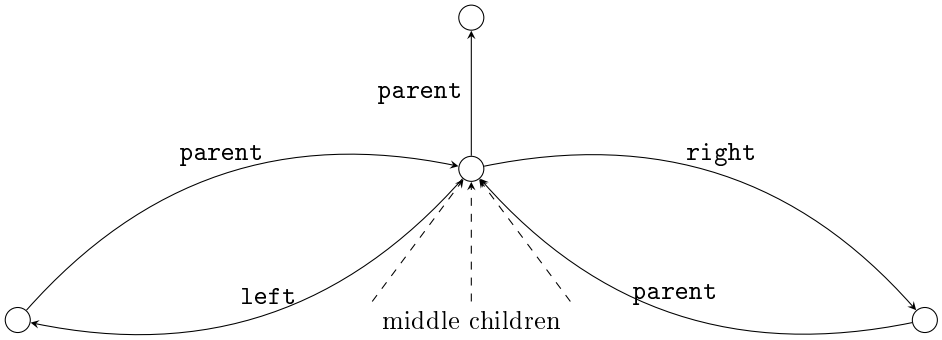
איך נבצע $(w, c) = \texttt{mincost}(v)$? נכניס את הערכים שבקשתות לתוך הצמתים שמהן הן יוצאות. כל צומת יזכור לא רק את הערך שלו, אלא גם את המינימום שלו בתוך כל תת־העץ שלו (בעץ ה־Splay). נבצע Splay ל־v, וא נסתכל תמיד על הערך שלו, וגם על המינימום של תת־העץ. בכל רגע, נדע אם להתקדם שמאלה או ימינה לפי המספרים האלה. בסוף, נגיע למינימום, ונעשה עליו Splay.

איך נעשה $\texttt{addcost}(v, c)$? כל מה שעשינו עד עכשיו לא מספיק בשביל addcost משיקולי סיבוכיות. במקום לכתוב את $c(v, p(v))$ בכל צומת, נכתוב בכל צומת את $\Delta c(v, p(v)) = c(v, p(v)) - c(p(v), p(p(v)))$. כמו כן, נשמור את $\Delta \min c$ במקום את $\min c$.

עכשיו אפשר לעשות $\texttt{addcost}(v, c)$ בצורה פשוטה: נשנה את העלות של v, ונתקן את העלות של הבן השמאלי שלו. המימוש של mincost ישתנה קצת, כי עכשיו צריך לחשב את הערכים האמיתיים בזמן הטיול למטה. כמו כן, עדיין אפשר לבצע את הסיבובים של ה־Splay, כי העלויות לא משתנות, אלא רק ההפרשים מהעלות של האבא, כיהאבא משתנה.

איך נבצע $\texttt{link}(v, w, c(v, w))$? כשמדברים על שרוכים, האפשרות היחידה לביצוע link היא לתלות שורש על זנב. את זה עושים בעזרת Catenate של עצי Splay. את

[1] מבצעים את ה־Splay הזה משיקולי פוטנציאל בחישוב הסיבוכיות.



איור 2.4: דוגמה לצומת עם ילדים וירטואליים.

$\text{cut}(v)$ עושים בעזרת Split של עצי Splay.
אלגוריתם 1.2 מתאר את ביצוע כל הפעולות במקרה הפרטי הזה.

2.3.2 המקרה הכללי

השלב הבא יהיה להפוך את המסלולים לעצים אמיתיים, כדי שלא נהיה מוגבלים.

כדי לממש את המקרה הכללי (שהעץ הוא לא בהכרח שרוך), נשתמש בעצים וירטואליים. עץ וירטואלי הוא עץ בינארי, שלכל צומת יש גם ילדים אמצעיים (כפי שניתן לראות באיור 2.4). הילדים באמצעיים מחוברים בצורה לוגית לצומת, והם מייצגים שורשים של עצים אחרים. קיבלנו מצב שבו בפועל יש לנו מסלולים (שרוכים) זרים, שמחוברים בקשתות לוגיות. כל מסלול מיוצג בעזרת עץ בינארי מאוזן, והעצים תלויים על הצומת שממנו המסלול יוצא. כל מסלול נקרא Solid Subtree בעץ הוירטואלי. ראה את הדוגמה באיורים 2.5 ו־2.6.

איך נייצג עצים וירטואליים? כל קודקוד ישמור מצביעים לאבא, ולבנים הימני והשמאלי. לא שומרים מצביעים לילדים האמצעיים (כפי שניתן לראות באיור 2.4). נתייחס לכל Solid Subtree כאל Splay Tree נפרד.

איך נייצג את העלויות? בדיוק כמו קודם, בתוך כל Solid Subtree. בשורש של כל Solid Subtree נייצג את העלות האבסולוטית (ללא תלות באבא).

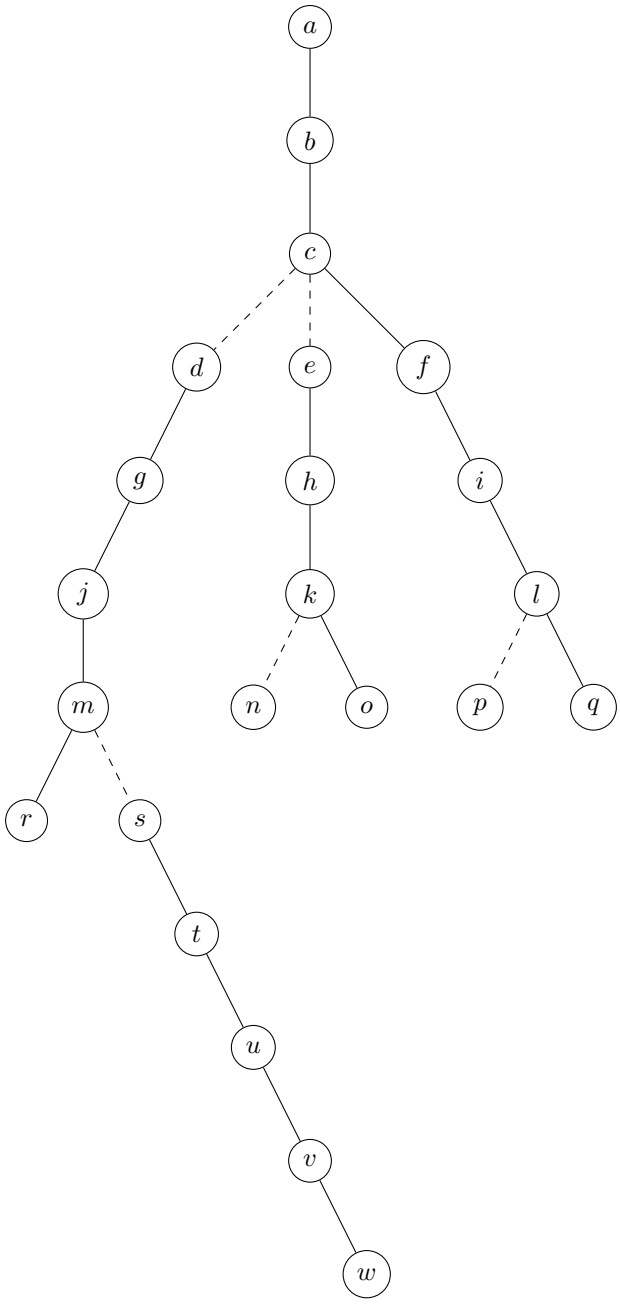
2.3.2.1 פעולות בעצים הוירטואליים

פעולת Splice אנחנו רוצים שקודקוד v יהיה על אותו המסלול (בעצים וירטואליים) כמו השורש (בעץ האמיתי). אזי אנחנו נחליף בינו לבין הבן השמאלי של אבא של v, כך ש־v יהיה הבן השמאלי של אבא שלו, ואם היה לאבא שלו בן שמאלי אחר, הוא יהיה עכשיו בן אמצעי. ניתן לראות המחשה לפעולה באיור 2.7, ופסאודו־קוד באלגוריתם 2.2.

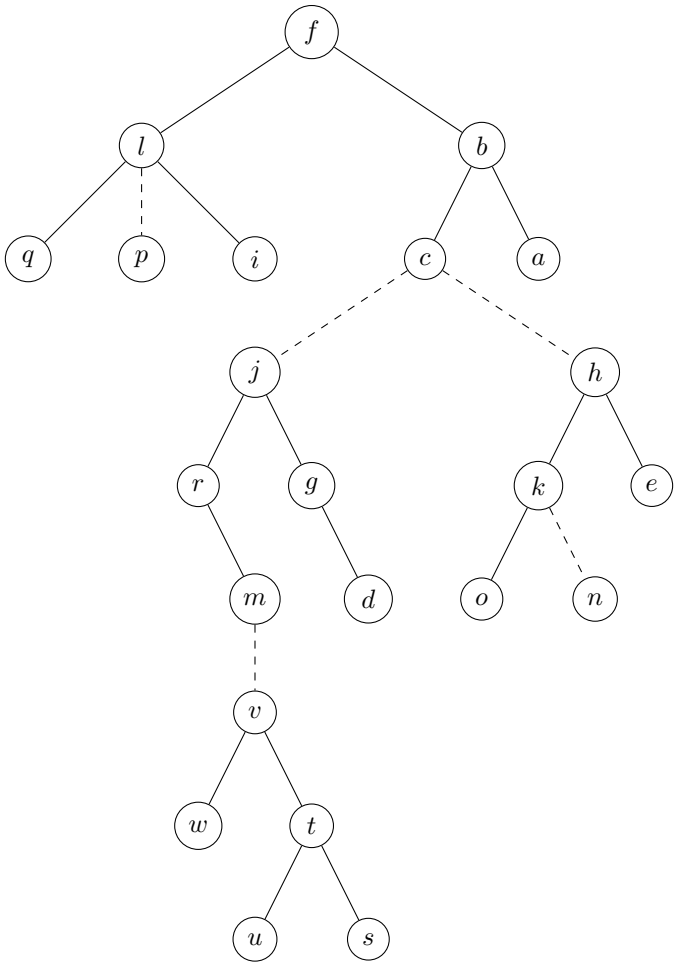
חשוב לשים לב שאפשר לעשות Splice בזמן קבוע. כמו כן, גם v וגם w הם שורשים (כי בוצע עליהם Splay ב־$\mathcal{O}(\log n)$), וכך אפשר לבצע את זה בזמן קבוע. כמו כן, w לא חייב להיות אבא של v בעץ האמיתי, אלא רק בעצים הוירטואליים.

אלגוריתם 1.2 מימוש פעולות לעצים דינאמיים כאשר כל עץ הוא שרוך

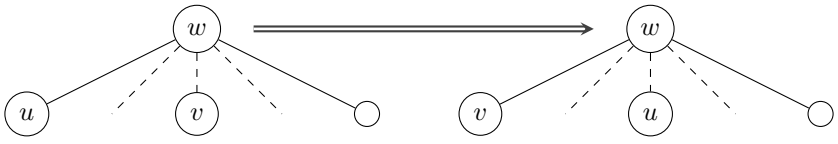
```
function MakeTree(v)
    Create a splay tree with a single node – v
end function
function FindRoot(v)
    Splay(v)
    w ← v
    while w.right ≠ nil do w ← w.right
    Splay(w)
    return w
end function
function MinCost(v)
    Splay(v)
    w ← v
    while w.Δmincost ≠ 0 do
        if w.Δmincost = w.left.Δmincost then w ← w.left
        else w ← w.right
    end while
    Splay(w)    ▷ Now w is the root, and therefore w.Δcost is the cost of w
    return (w, w.Δcost)
end function
function AddCost(v, c)
    Splay(v)
    v.Δcost ← v.Δcost + c
    v.left.Δcost ← v.left.Δcost + c
end function
function Link(v, w, c(v, w))
    Splay(v) ; Splay(w)
    Catenate(v, w)
    v.Δcost ← c(v, w) − w.Δcost
    v.Δmincost ← v.Δmincost − w.Δmincost
end function
function Cut(v)
    Splay(v)
    w ← v.right
    Cut(v)
    w.Δcost ← w.Δcost + v.Δcost
    w.Δmincost ← w.Δmincost + v.Δmincost
end function
```



איור 2.5: פירוק של עץ דינאמי להרבה שרוכים



איור 2.6: הפירוק לעצים וירטואליים של העץ הדינאמי מאיור 2.5



איור 2.7: פעולת Splice של w

אלגוריתם 2.2 פעולת Splice על w

```
function SPLICE(v)
    w ← v.parent
    u ← w.left
    w.left ← w
    u.Δcost ← u.Δcost + w.Δcost
    v.Δcost ← v.Δcost − w.Δcost
    w.Δmin ← max{0, v.Δmin − v.Δcost, w.right.Δmin − w.right.Δcost}
end function
```

פעולת Splay בעץ וירטואלי איך נעשה Splay לצומת בתוך העץ הוירטואלי (ולא רק בתוך ה־Solid Subtree)?

1. עושים Splay ל־x בתוך כל Solid Subtree. מדלגים כל פעם לעץ הקשיר הבא, ועוד פעם ועוד פעם. אחרי שעשינו את זה, המסלול מ־x לשורש הוא מסלול של קשתות מקווקוות (של ילדים אמצעיים) בין שורשים של Solid Subtrees.
2. עושים Splice על כל אחת מהקשתות המקווקוות.
3. עושים Splay ל־x בתוך ה־Solid Subtree שהתקבל. עכשיו x הוא השורש של העץ הוירטואלי.

2.3.2.2 פעולות של העצים הדינאמיים

למעשה, המימוש של העצים הדינאמיים במקרה הכללי לא השתנה בצורה מהותית מהמימוש לעצים דינאמיים במקרה שבו כל העצים הם שרוכים. הדבר היחיד שצריך לזכור הוא שמדובר עכשיו בעצים וירטואליים, שאינם קשירים. אולם ביצוע Splay לצומת בעץ וירטואלי מקשרת את כל המסלול ממנו עד לשורש שלו בתוך Solid Subtree, ולכן אין צורך לשנות כמעט וכלום.

המימושים לפעולות העצים הדינאמיים מופיעים באלגוריתם 3.2.

2.4 ניתוח

נרצה לראות שכל הפעולות לוקחות $\mathcal{O}(\log n)$, כפי שהובטח. לשם כך, צריך להראות ש־Splay בעץ וירטואלי לוקח בממוצע (Amortized) $\mathcal{O}(\log n)$. אם זה נכון, קל לראות שכל הפעולות של העצים הדינאמיים רצות בזמן לוגריתמי, משום שהן עושות מספר קבוע של פעולות Splay, ועוד מספר קבוע של פעולות בזמן קבוע.

למת הגישה לא תקפה במקרה הזה, כי אנחנו לא מדברים על Splay Tree רגיל, אלא על עץ וירטואלי. למת הגישה (למה 3) אמרה שזמן Amortized של פעולת Splay (בעץ Splay) הוא:

$$3 \cdot (r(t) - r(x)) + 1 = 3 \cdot \log \frac{s(t)}{s(x)} + 1$$

כאשר r הוא השורש, ו־x הוא הצומת שעליו מבצעים Splay.

אלגוריתם 3.2 מימוש פעולות לעצים דינאמיים במקרה הכללי. כאשר מבצעים Splay, הכוונה היא ל־Splay בעצים הוירטואליים.

```
function MakeTree(v)
    Create a splay tree with a single node – v
end function
function FindRoot(v)
    Splay(v)
    w ← v
    while w.right ≠ nil do w ← w.right
    Splay(w)
    return w
end function
function MinCost(v)
    Splay(v)
    w ← v
    while w.Δmincost ≠ 0 do
        if w.Δmincost = w.left.Δmincost then w ← w.left
        else w ← w.right
    end while
    Splay(w)     ▷ Now w is the root, and therefore w.Δcost is the cost of w
    return (w, w.Δcost)
end function
function AddCost(v, c)
    Splay(v)
    v.Δcost ← v.Δcost + c
    v.left.Δcost ← v.left.Δcost + c
end function
function Link(v, w, c(v, w))
    Splay(v) ; Splay(w)
    v.parent ← w                                  ▷ Make v a middle child of w
    v.Δcost ← c(v, w) − w.Δcost
    v.Δmincost ← v.Δmincost − w.Δmincost
end function
function Cut(v)
    Splay(v)
    w ← v.right
    Cut(v)
    w.Δcost ← w.Δcost + v.Δcost
    w.Δmincost ← w.Δmincost + v.Δmincost
end function
```

נשנה את הגדרת הגוזל $s(x)$ כך שיהיה מספר הצאצאים הכולל של x בעץ הוירטואלי. הפוטנציאל יהיה c פעמים סכום הדרגות, עבור c קבוע כלשהו. c יעזור לנו לאפשר יותר משינוי אחד בכל סיבוב Splay. נשים לב שפונקציית המשקל $w(x)$ נותנת את המשקל 1 לכל צומת בעץ.

נשים לב שלמת הגישה (הרגילה של עצי Splay) עדיין תקפה. לפי למת הגישה, ה־Splay הראשון לקח $3 \cdot \log \frac{s(T_x)}{s(x)} + 1$. השני — $3 \cdot \log \frac{s(T_1)}{s(x_1)} + 1$. ה־$k$ — $3 \cdot \log \frac{s(T_k)}{s(x_k)} + 1$. נסכום את כולם ביחד:

$$\begin{aligned}
\text{Amortized Time} &= 3 \cdot \log \frac{s(T_k)}{s(x_k)} + \cdots + 3 \cdot \log \frac{s(T_1)}{s(x_1)} + 3 \cdot \log \frac{s(T_x)}{s(x)} + k \\
&= 3 \cdot \log s(T_k) - \underbrace{3 \cdot \log s(x_k) + 3 \cdot \log s(T_{k-1})}_{\leq 0} + \underbrace{\cdots}_{\leq 0} \\
&\quad \underbrace{-3 \cdot \log s(x_1) + 3 \cdot \log s(T_x)}_{\leq 0} - 3 \cdot \log s(x) + k \\
&\leq 3 \cdot \log s(T_k) - 3 \cdot \log s(x) + k
\end{aligned}$$

כאשר k הוא מספר פעולות ה־Splay שעשינו ב־Solid Subtrees. לכן, שלב 1 לוקח $3 \cdot \log \frac{s(T_k)}{s(x)} + k = 3 \cdot \log n + k$. שלב 2 עושה $k-1$ פעולות בזמן קבוע.

למה 15 (הכללה של למת הגישה): לכל קבוע c, הזמן המפוצץ (*Amortized*) של פעולת *Splay* הוא

$$3 \cdot c \cdot \log \frac{s(T)}{s(x)} + 1 + (c-1) \cdot (k-1)$$

כאשר k הוא אורך מסלול ה־*Splay*.

הערה 16: למת הגישה היא מקרה פרטי של למה זו, כאשר $c = 1$.

לא נוכיח את ההכללה של למת הגישה.

מההכללה של למת הגישה, שלב 3 לוקח $3 \cdot c \log n + 1 - (c-1) \cdot (k-1)$.

בסך הכל, פעולת Splay של עץ וירטולי לוקחת:

$$3 \cdot \log n + k + (k-1) + 3 \cdot c \cdot \log n + 1 - (c-1) \cdot (k-1)$$

אם נבחר c מספיק גדול, הקבוע האדיטיבי k יתבטל.

פרק 3

זרימה מקסימלית

המטרה שלנו באלגוריתמי זרימה תהיה למצוא זרימה מקסימלית מ־s ל־t בגרף מכוון בזמן $\mathcal{O}(m \cdot n \cdot \log n)$. עם קצת מאמץ, נגיע גם לזמן $\mathcal{O}\left(m \cdot n \cdot \log {n^2}/{m}\right)$. אלגוריתמי הזרימה הבסיסיים (שרואים בתואר הראשון) רצים בזמן $\mathcal{O}\left(m \cdot n^2\right)$.

3.1 הגדרות

3.1.1 זרימה

הקלט שלנו יהיה גרף מכוון $G = (V, E)$, ונתונים שני צמתים $s, t \in V$, $s \neq t$. לכל קשת $(v, w) \in E$ בגרף, יש קיבולת, המסומנת ב־$u(v, w)$.

הגדרה 17 (זרימה): פונקציה $f : E \to \mathbb{R}$ תקרא זרימה (*flow*) אם היא מקיימת את התנאים הבאים:

$$\forall (v, w) \in E. 0 \leq f(v, w) \leq u(v, w) \tag{3.1}$$

$$\forall v \in V \setminus \{s, t\} . \sum_{(v,w)\in E} f(v, w) - \sum_{(w,v)\in E} f(w, v) = 0 \tag{3.2}$$

הגדרה 18 (הגדרה שקולה לזרימה): בהינתן גרף $G = (V, E)$, לכל קשת $(v, w) \in E$, אם $(w, v) \notin E$, אז נוסיף את הקשת (w, v) עם $u(w, v) = 0$.
אזי הפונקציה $f : E \to \mathbb{R}$ תקרא זרימה אם היא מקיימת:

1. אנטי־סימטריות:

$$\forall (v, w) \in E. f(v, w) = -f(w, v)$$

2. קיבול:

$$\forall (v,w) \in E. f(v,w) \leq u(v,w)$$

הערה 19: הפעם לא דורשים ש־f תהיה אי־שלילית, משום שמהאנטי־סימטריות, f עשוייה לקבל ערכים שליליים.

3. שימור הזרימה:

$$\forall v \in V \setminus \{s,t\} . \sum_w f(v,w) = 0$$

למה 20: הגדרות *17* ו־*18* שקולות.

הוכחה: אם f היא זרימה לפי הגדרה 17, אזי נגדיר את f' על ידי

$$f'(v,w) = f(v,w) - f(w,v)$$

אזי f' היא זרימה לפי הגדרה 18.
אם f היא זרימה לפי הגדרה 18, אזי נגדיר את f' על ידי

$$f'(v,w) = \max\{0, f(v,w)\}$$

אזי f היא זרימה לפי הגדרה 17. ■

הגדרה 21 (ערך של זרימה): תהי f זרימה. הערך של הזרימה f הוא:

$$|f| = \sum_w f(s,w)$$

עבור צומת w כלשהו.

הערה 22: קל לראות כי:

$$|f| = \sum_w f(w,t)$$

בעיית הזרימה המקסימלית מחפשת זרימה f שהערך שלה מקסימלי.

3.1.2 חלוקות s–t

נחלק את קבוצת הצמתים V לשתי קבוצות שונות, X ו־X', כך ש־$X \cup X' = V$, $X \cap X' = \emptyset$, $s \in X$ ו־$t \in X'$.
עבור זרימה f כלשהי:

$$f(X,X') = \sum_{\substack{v \in X \\ w \in X'}} f(v,w) = \sum_{\substack{v \in X \\ w \in V}} f(v,w) - \sum_{\substack{v \in X \\ w \in X}} f(v,w) = |f| - 0 = |f|$$

לכן:

$$|f| \leq \mathrm{cap}\left(X, X'\right) = \sum_{\substack{v \in X \\ w \in X'}} u\left(v, w\right)$$

לכן, נסיק כי הערך של הזרימה המקסימלית הוא לכל היותר קיבול של חתך כלשהו, ולכן הערך של הזרימה המקסימלית הוא לכל היותר הקיבול המינימלי של חתך.

3.1.3 רשת שיורית

הגדרה 23 (קיבול שיורי): הקיבול השיורי (*Residual Capacity*) של זרימה f הוא פונקציה $r : E \to \mathbb{R}$ המקיימת:

$$r\left(v, w\right) = u\left(v, w\right) - f\left(v, w\right)$$

האינטואיציה להגדרה היא שניתן להזרים $r\left(v, w\right)$ יותר בקשת $\left(v, w\right)$ באמצעות הגדלת $f\left(v, w\right)$ והקטנת $f\left(w, v\right)$.

הגדרה 24 (רשת שיורות): מהקיבול השיורי ניתן להגדיר את הרשת השיורית (*Residual Network*), המתקבלת מקבוצת הקשתות שהקיבול שלהן שונה מ־0.
במילים אחרות, הרשת השיורית היא הגרף $G_R = (V, R)$ כאשר $R = \{(v, w) \in E \mid r\left(v, w\right) \neq 0\}$.

הגדרה 25 (מסילה משפרת): מסילה משפרת או מסלול משפר (או *Augmenting Path*) הוא מסלול ברשת השיורית מ־s ל־t. המסלול נקרא משפר משום שהוא מאפשר שיפור של הזרימה.
בצורה יותר פורמלית: אם $p \in R$ מסלול מ־s ל־t ברשת השיורית, אזי ניתן להגדיל את הזרימה f על כל קשת במסלול p ב־$\min_{(v,w)\in p} r\left(v, w\right)$.

משפט 26: הטענות הבאות שקולות:

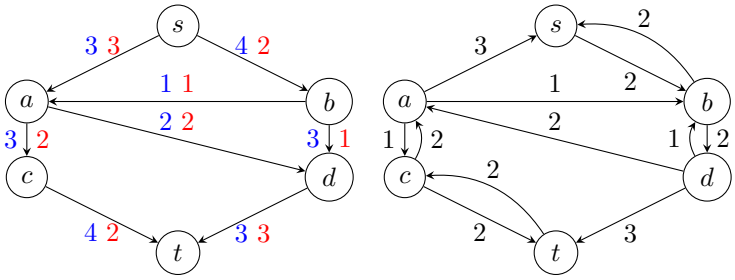
1. f היא זרימה מקסימלית.
2. אין מסילה משפרת ב־R.
3. $|f| = cap\left(X, X'\right)$ עבור X ו־X' המקיימים $X \cup X' = V$, $X \cap X' = \emptyset$, $s \in X$ ו־$t \in X'$.

הוכחה: נוכיח את השקילויות:

1 ו־3 שקולים ראינו קודם.

2 גורר את 3 תהי X קבוצת כל הצמתים הנגישים מ־s ב־R. לפי ההנחה שאין מסילה משפרת, $t \notin R$. נגדיר: $X' = V \setminus X$. אזי (X, X') היא חלוקת s–t. מכיוון שאין קשתות מ־X ל־X':

$$|f| = f\left(X, X'\right) = \sum_{\substack{v \in X \\ w \in X'}} f\left(v, w\right) = \sum_{\substack{v \in X \\ w \in X'}} u\left(v, w\right) = \mathrm{cap}\left(X, X'\right)$$



איור 3.1: זרימה חוסמת שאינה מקסימלית. הגרף השמאלי הוא הגרף הרגיל. המספרים הכחולים הם קיבולי הקשתות, והמספרים האדומים הם הזרימה על הקשתות. הגרף הימני הוא הרשת השיורית, והמספרים הם הקיבול השיורי של כל קשת (לפי כיווני החצים). קל לראות שעל כל מסלול מ־s ל־t יש קשת רוויה (ולכן הזרימה חוסמת), אך קיים מסלול מ־s ל־t ברשת השיורית, ולכן הזרימה אינה מקסימלית.

3 גורר את 2 מכיוון ש־$|f| = \text{cap}(X, X')$, נקבל כי על כל קשת (v, w) כך ש־$v \in X$ ו־$w \in X'$, מתקיים $f(v, w) = u(v, w)$. לכן, אף אחת מהקשתות של החתך לא קיימות ברשת השיורית, ולכן אין מסילה משפרת ב־R.

שאר כיווני השקילות נובעים ממה שכבר הוכחנו. ■

3.2 האלגוריתם של Dinic

צריך למצוא מסילה משפרת ברשת השיורית, ולעצור כש־s ו־t אינם מחוברים יותר ברשת השיורית. צריך להיזהר כשמפעילים את זה, כי לא מובטח שהאלגוריתם יעצר.

מסתבר שצריך לקחת את המסילה המשפרת הקצרה ביותר (לפי מספר הקשתות). זה לא יעיל, אבל עובד, ובזמן פולינומי ($\mathcal{O}(m^2 \cdot n)$). האלגוריתם נקרא Edmonds and Karp. האלגוריתם של Dinic יותר יעיל.

הגדרה 27 (זרימה חוסמת): זרימה f היא חוסמת אם כל מסלול מ־s ל־t מכיל קשת רוויה.

הערה 28: זרימה חוסמת אינה בהכרח מקסימלית. ראה דוגמה באיור 3.1.

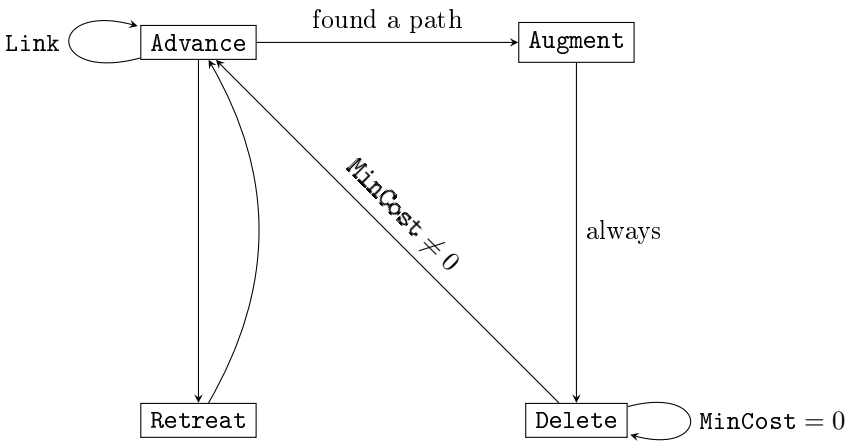
הגדרה 29 (רמה של צומת): יהי v צומת כלשהו. נסמן ב־$\texttt{level}(v)$ את האורך של המסלול הקצר ביותר מ־s ל־v.

אפשר למצוא את $\texttt{level}(v)$ לכל $v \in V$ על ידי הפעלת BFS שמתחיל מ־s.

לפי האלגוריתם של Dinic, צריך להגדיל את הזרימה בדרך הבאה: נסתכל על L, שהוא תת־גרף של הרשת השיורית, כך שכל קשת $(v, w) \in L$ מקיימת $\texttt{level}(w) = \texttt{level}(v) + 1$. האלגוריתם ימצא זרימה חוסמת f' ב־L, ויגדיל את f בעזרת f'.

3.2.1 ניתוח סיבוכיות

משפט 30: f היא זרימה מקסימלית אחרי לכל היותר $n - 1$ חישובי זרימה חוסמת.



איור 3.2: גרף המצבים של האלגוריתם של Dinic עם עצים דינאמיים

הוכחה: כל קשת ב־L היא היא או קשת ב־G או קשת הפוכה של G.
נסתכל על המסלול הקצר ביותר מ־s ל־t ב־L. הרמה ב־R גדלה בלכל היותר 1, אבל לא יכולה לגדול תמיד ב־1 בדיוק. ■

לכן, אם נשתמש ב־DFS פשוט, נקבל $\mathcal{O}(n \cdot m)$, ולכן סך זמן הריצה של האלגוריתם יהיה $\mathcal{O}(n^2 \cdot m)$.
אפשר להשיג חסם טוב יותר באמצעות שימוש בעצים דינאמיים.

3.2.2 עצים דינאמיים

האלגוריתם של Dinic מוצא לכל היותר $n-1$ זרימות חוסמות ברשת השיורית. בעזרת DFS אפשר לעשות את זה ב־$\mathcal{O}(m \cdot n)$. התהליך הזה לא יעיל, כי הוא מאבד אינפורמציה. בכל פעם שזורקים קשת רווייה, נשארים לפחות עם שתי מסילות שיוריות שאינן רווייות, ואולי אפשר להשתמש במידע הזה. נמצא זרימה חוסמת מהר יותר בעזרת עצים דינאמיים.
נתחזק אוסף של עצים (תת־גרף של L), כך שלכל צומת v יש לכל היותר קשת אחת שאינה רוויה שיוצאת ממנו. המשקל של כל קשת יהיה הקיבול השיורי שלה. האלגוריתם מתואר באלגוריתם 1.3. האלגוריתם מתחיל מכך שכל צומת הוא עץ נפרד (Singleton) בגרף הדינאמי. ההנחה היא שבעץ הדינאמי, `MinCost` מחזיר את הצומת הכי קרוב לשורש (t באלגוריתם הזה).
ניתוח:

1. זמן הריצה פרופורציוני לזמן שלוקח לעשות את הפעולות על העץ הדינמי ועוד $\mathcal{O}(m)$ עבור ניקוי הקשתות שאף פעם לא נכנסו לעצים.
2. בין כל שתי פעולות `Link` ו־`Cut` מבצעים $\mathcal{O}(1)$ פעולות אחרות.

אלגוריתם 1.3 מציאת זרימה מקסימלית לפי Dinic בעצים דינאמיים

```
1: procedure ADVANCE
2:     v ← FINDROOT(s)
3:     if v = t then AUGMENT
4:     if there is no outgoing edge from v then RETREAT
5:     choose an edge (v, w) going out of v
6:     LINK(v, w, r(v, w)) ADVANCE
7: end procedure

8: procedure AUGMENT
9:     v, c ← MINCOST(s)
10:    ADDCOST(s, −c)
11:    DELETE
12: end procedure

13: procedure DELETE                ▷ delete the edge (v, p(v)) from the graph
14:    f(v, p(v)) ← u(v, p(v))
15:    CUT(v)
16:    v, c ← MINCOST(s)
17:    if c = 0 then DELETE
18:    else ADVANCE
19: end procedure

20: procedure RETREAT               ▷ all paths from v to t are blocked
21:    if v = s then halt           ▷ the algorithm is terminated
22:    for all edge (u, v) in the graph do   ▷ delete evert edge (u, v) from the
    graph
23:        if v ≠ p(u) then f(u, v) ← 0
24:        else
25:            u, Δ ← MINCOST(u)
26:            f(u, v) ← u(u, v) − Δ
27:            CUT(u)
28:        end if
29:    end for
30:    ADVANCE
31: end procedure
```

3. מספר פעולות ה־`Link` ו־`Cut` הוא $\mathcal{O}(m)$, כי כל קשת נכנסת ליער לכל היותר פעם אחת בלבד.

מסקנה 31: זמן הריצה של *Dinic* על עץ דינאמי הוא $\mathcal{O}(n \cdot m \cdot \log n)$, כי מציאת זרימה חוסמת לוקחת $\mathcal{O}(m \cdot \log n)$.

פרק 4

זרימה מקסימלית בשיטת Push/Relabel

4.1 Distance Labels

הגדרה 32 (Distance Label): יהי גרף מכוון $G = (V, E)$ עם קיבול לקשתות (כמו קודם), ושני צמתים s ו־t. פונקציית *Distance Label* מקיימת $d(t) = 0$, $d(s) = n$, ולכל קשת שיורית (v, w) כך ש־$r(v, w) > 0$, $d(v) \leq d(w) + 1$.

למה 33: $d(v)$ הוא חסם תחתון לאורך של המסלול הקצר ביותר מ־v ל־t ברשת השיורית.

הוכחה: יהי $(v, v_1, v_2, \ldots, v_k = t)$ המסלול הקצר ביותר מ־v ל־t ברשת השיורית. אזי לכל i, $d(v_i) \leq d(v_{i+1}) + 1$. לכן:

$$d(v) \leq d(v_1) + 1 \leq d(v_2) + 2 \leq \cdots \leq d(v_k) + k = d(t) + k = k$$

■

הערה 34: כדי ש־Distance Label יהיה חוקי, s ו־t לא יכולים להיות מחוברים. אם s ו־t מחוברים (ברשת השיורית), אז קל לראות את הסתירה מהלמה.

הגדרה 35 (קשת Admissible): קשת (v, w) ברשת השיורית היא *Admissible* אם $d(v) = d(w) + 1$.

4.2 Preflow

הגדרה 36 (Preflow): יהי רף מכוון $G = (V, E)$ עם קיבול לקשתות (כמו קודם), ושני צמתים s ו־t. *Preflow* היא פונקציה f על הקשתות ב־E שמקיימת:

1. $f(v, w) = -f(w, v)$.
2. $f(v, w) \leq u(v, w)$.

אלגוריתם 1.4 Push/Relabel

```
1: while there is an active vertex v do
2:     Pick an active vertex v
3:     PushRelabel(v)
4: end while

5: procedure PushRelabel(v)
6:     if there is an admissible arc (v,w) then push δ = min{e(v), r(v,w)}
       flow from v to w
7:     else relabel d(v) ← min{d(w) + 1 | r(v,w) > 0}
8: end procedure
```

3. לכל w שאינו s או t:

$$\sum_v f(v,w) \geq 0$$

נסמן:

$$e(w) = \sum_v f(v,w)$$

נקרא ל־$e(w)$ ה־$excess$ של w.

הגדרה 37 (צומת פעיל): צומת v כך ש־$e(v) > 0$ נקרא פעיל.

הערה 38: נשים לב כי $e(t) \geq 0$ ו־$e(s) \leq 0$.

4.3 אלגוריתם Push/Relabel

האלגוריתם מתואר באלגוריתם 1.4.

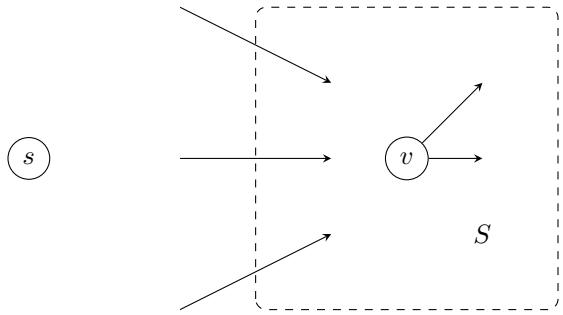
4.3.1 נכונות

למה 39: המקור s נגיש מכל צומת פעיל ברשת השיורית.

הוכחה: נניח בשלילה שהמקור לא נגיש מהצומת v ברשת השיורית. אזי איור 4.1 מתאר את המצב הנוכחי של הרשת.
מכיוון שאין מסלול מ־v ל־s ברשת השיורית, אזי אין זרימה מ־s ל־v. אבל אז לא ייתכן ש־v פעיל, בסתירה! ■

מסקנה 40: תמיד יש קשת שיורית שיוצאת מצומת פעיל.

מסקנה 41: האלגוריתם *Push/Relabel* מוגדר היטב, כלומר אפשר לבצע *Push/Relabel* כל עוד יש צומת פעיל, וה־min ב־*relabel* הוא לא על קבוצה ריקה.



איור 4.1: תיאור המצב כאשר s לא נגיש מהצומת הפעיל v ברשת השיורית

למה 42: ה־*Distance Labels* רק גדלים, והחוקיות שלהם נשמרת.

הוכחה: באינדוקציה על מספר פעולות ה־Push וה־Relabel.
אם עושים Relabel, ה־Distance Label נשמר לפי ההגדרה.
אם עושים Push מ־v ל־w, אזי $d(v) = d(w) + 1$, ולכן גם אם הקשת (w, v) מתווספת לרשת השיורית, d הוא עדיין Distance Label חוקי. ■

למה 43: כאשר (ואם) האלגוריתם עוצר, ה־*Preflow* הוא זרימה מקסימלית.

הוכחה: זוהי זרימה מכיוון שאין צמתים פעילים.
הזרימה מקסימלית מכיוון ש־t לא נגיש מ־s ברשת השיורית (כי $d(s) = n$, ו־d הוא חוקי). ■

4.3.2 ניתוח סיבוכיות

תזכורת 44: נשים לב לכך ש־$d(v)$ גדל כאשר עושים *Relabel* ל־v.

למה 45: $d(v) \leq 2 \cdot n - 1$.

הוכחה: יהי מסלול $(v, v_1, v_2, \ldots, v_k = s)$ מ־v ל־s ברשת השיורית. אזי $d(v_i) \leq d(v_{i+1}) + 1$ לכל i. כמו כן, ברור כי $k \leq n - 1$. לכן:

$$d(v) \leq d(v_1) + 1 \leq d(v_2) + 2 \leq \cdots \leq d(v_k) + k \leq d(s) + n - 1 = 2 \cdot n - 1$$

■

מסקנה 46: מספר פעולות ה־*Relabel* הוא $(2 \cdot n - 1) \cdot (n - 2) \leq 2 \cdot n^2$.

הוכחה: יש לכל היותר $2 \cdot n - 1$ פעולות Relabel לכל צומת שאינו s או t. ■

הגדרה 47 (Push מרווה): נקרא לפעולת Push מרווה (*Saturating*) אם $\min\{e(v), r(v, w)\} = r(v, w)$.

למה 48: מספר ה־*Push*־ים המרווים הוא לכל היותר $2 \cdot n \cdot m$.

הוכחה: נניח שזה עתה ביצענו Push מרווה על (v, w). לפני שנוכל לבצע עוד Push מרווה על (v, w), נצטרך לדחוף זרימה מ־w ל־v.

לכן, $d(w)$ חייב לגדול בלפחות 2. מכיוון ש־$d(w) \leq 2 \cdot n - 1$, זה יכול לקרות לכל היותר n פעמים. לכן, על כל קשת יש לכל היותר n Push־ים מרווים (בכל כיוון). ■

למה 49: מספר ה־*Push*־ים שאינם מרווים הוא לכל היותר $4 \cdot n^2 \cdot m$.

הוכחה: נגדיר:

$$\Phi = \sum_{v \text{active}} d(v)$$

אז:

- Φ קטן (לפחות ב־1) בכל Push שאינו מרווה.
- Φ גדול לכל היותר ב־$2 \cdot n - 1$ בכל Push מרווה. מלמה 48, Φ גדל בלכל היותר $(2 \cdot n - 1) \cdot 2 \cdot n \cdot m$ בכל ה־Push־ים המרווים.
- Φ גדל בכל Relabel, בסך הכל, פחות מ־$(2 \cdot n - 1) \cdot (n - 2)$. ■

4.3.3 מימוש

נתחזק רשימה של צמתים פעילים, כך שיהיה קל למצוא צומת פעיל.

כמו כן, בהינתן צומת פעיל v, נצטרך לקבוע אם יש קשת Admissible שיוצאת ממנו, כדי לדעת אם לעשות Push או Relabel. נשמור לכל צומת את כל הקשתות היוצאות ממנו ברשימה מקושרת, וכל פעם נסתכל על הקשת הנוכחית. אם הקשת הנוכחית (v, w) היא Admissible, אז נבצע Push עליה (וגם נעדכן את הרשימה של הצמתים הפעילים). אחרת, נקדם את המצביע של הקשת הנוכחית ברשימה (לפי הסדר). אם הגענו לסוף הרשימה, נבצע Relabel ל־v, ונקבע שהקשת הנוכחית שלו היא הראשונה ברשימה.

למה 50: כשמבצעים *Relabel* ל־v, אין קשת *Admissible* מהצורה (v, w).

הוכחה: אחרי שסרקנו את הקשת (v, w), או ש־(v, w) יצאה מהרשת השיורית, או ש־$d(v) \leq d(w)$.

אם $d(v) \leq d(w)$, אז זה חייב להמשיך להתקיים עד ה־Relabel של v, שכן $d(v)$ לא משתנה עד ל־Relabel, ו־$d(w)$ יכול רק לגדול.

אם (v, w) נכנסה שוב לרשת השיורית אחרי שסרקנו אותה פעם אחרונה, אז כשהיא נכנסה שוב לרשת השיורית התקיים $d(w) = d(v) + 1$, ולכן $d(v) \leq d(w)$, וזה כמו במקרה הקודם. ■

למה 51: הזמן הכולל המושקע ב־v בין שתי פעולות *Relabel* הוא Δ_v ועוד $\mathcal{O}(1)$ לכל *Push* שיוצא מ־v, כאשר Δ_v הוא מספר הקשתות היוצאות מ־v.

מסקנה 52: מכיוון שמבצעים *Relabel* ל־v לכל היותר $2 \cdot n - 1$ פעמים, נקבל שהעבודה הכוללת המושקעת ב־v היא $\mathcal{O}(n \cdot \Delta_v)$ ועוד $\mathcal{O}(1)$ לכל *Push* יוצא מ־v.

אם נסכום על כל הצמתים, נקבל שהזמן הכולל הוא $\mathcal{O}(m \cdot n)$ ועוד מספר פעולות ה־*Push* הכולל. לכן, זמן הריצה הכולל הוא $\mathcal{O}\left(n^2 \cdot m\right)$.

אלגוריתם 2.4 פעולת Discharge על הצומת v

```
1: procedure DISCHARGE(v)
2:     while v is active and has not been relabeled do PUSHRELABEL(v)
3:     if v has been relabeled then add v to the end of Q
4: end procedure
```

4.4 הקטנת מספר ה־Push־ים הלא מרווים

נתחזק רשימה של כל הצמתים הפעילים כתור FIFO, נקרא לו Q. נבצע פעולת Discharge על הצומת הראשון ב־Q. ראה אלגוריתם 2.4.

ניתן להסתכל על האלגוריתם לפי המעברים שלו. המעבר הראשון (Pass 1) מסתיים כאשר מסיימים לבצע Discharge על כל הצמתים שהיו ב־Q באתחול. המעבר ה־i (Pass i) מסתיים כאשר מסיימים לבצע Discharge על כל הצמתים שנכנסו ל־Q במהלך המעבר ה־$i-1$.

אנליזה נשים לב שהחסם $\mathcal{O}\left(n^2 \cdot m\right)$ עדיין תופס. כמה מעברים אנחנו עושים? נגדיר:

$$\Phi = \max_{\text{active } v} d\left(v\right)$$

1. אם האלגוריתם לא מבצע Relabel במהלך מעבר, אזי Φ קטן בלפחות 1, כי כל צומת פעיל בתחילת המעבר העביר את כל ה־excess שלו לצומת w עם $d\left(w\right) < d\left(v\right)$.
2. אם האלגוריתם מבצע Relabel במהלך מעבר, אזי Φ גדל בלכל היותר בגדילה המקסימלית של Distance Label שגדל.

יש לכל היותר $\mathcal{O}\left(n^2\right)$ מעברים מהסוג השני, והם מגדילים את Φ בלכל היותר $\mathcal{O}\left(n^2\right)$, ולכן יש לכל היותר $\mathcal{O}\left(n^2\right)$ מעברים מהסוג הראשון. כלומר, יש $\mathcal{O}\left(n^2\right)$ מעברים בסך הכל.

בכל מעבר יש לכל היותר Push לא מרווה אחד. לכן, יש לכל היותר $\mathcal{O}\left(n^3\right)$ פעולות Push לא מרוות, ולכן זמן הריצה הכולל של האלגוריתם הוא $\mathcal{O}\left(n^3\right)$.

4.5 עצים דינאמיים

ראינו איך למצוא זרימה חוסמת עם עצים דינאמיים ב־$\mathcal{O}\left(m \cdot \log n\right)$. עם הרעיון של Push/Relabel, נראה איך אפשר לשפר את מציאת הזרימה החוסמת ל־$\mathcal{O}\left(m \cdot \log \frac{n}{m^2}\right)$.

נתחזר יער דינאמי של חלק מהקשתות הנוכחית שהן Admissible. הצמתים הפעילים יהיו בין השורשים של העצים ביער. באופן כללי, האלגוריתם יהיה כמעט אותו הדבר: יהיה לנו תור FIFO של צמתים פעילים Q, וכל פעם נבצע Discharge על הצומת הראשון ב־Q. ההבדל יהיה שה־Discharge לא יקרא ל־PushRelabel, אלא ל־TreepushRelabel, שמבצע Tree Push (ששונה מ־Push רגיל) ו־Relabel (רגיל). תיאור של Tree Push ניתן למצוא באלגוריתם 3.4.

אלגוריתם 3.4 ביצוע Tree Push על הקשת (v, w)

```
1: if (v, w) is admissible then
2:     LINK(v, w, r_f(v, w))
3:     repeat
4:         v, c ← FINDMIN(v)
5:         c ← min{c, e(v)}
6:         ADDCOST(v, −c)
7:         repeat
8:             u, c ← FINDMIN(v)
9:             if c = 0 then CUT(u)
10:        until c ≠ 0
11:    until e(v) = 0 or v is a root
12: else
13:    if (v, w) is not the last edge then advance the current edge
14:    else
15:        for all child u of v do CUT(u)
16:        RELABEL(v)
17:    end if
18: end if
```

4.5.1 אנליזה

אנחנו מבצעים $\mathcal{O}(1)$ עבודה יחד עם פעולת `Link` או `Cut`, או שאנחנו מקדמים את הקשת הנוכחית.

כמה פעולות `Cut` יש לנו? $\mathcal{O}(m \cdot n)$ (כי כל אחד מחוייב על חשבון Push מרווה או Relabel). לכן, יש גם $\mathcal{O}(m \cdot n)$ פעולות `Link` (אחרת, אי אפשר לבצע `Cut` כל כך הרבה פעמים).

כמה פעמים `TreepushRelabel` מקדם את הקשת הנוכחית? $\mathcal{O}(m \cdot n)$.

לכן, האלגוריתם כולל $\mathcal{O}(m \cdot n)$ פעולות של עצים דינאמיים, ולכן זמן הריצה הכולל שלו הוא $\mathcal{O}(m \cdot n \cdot \log n)$.

4.5.2 שיפור לאנליזה

נשים לב שבאלגוריתם שלנו לא באמת השתמשנו בעובדה ש־Q הוא תור. כל רשימה הייתה משיגה את המטרה במימוש שראינו. נוכל לנצל את העובדה ש־Q הוא תור כדי להקטין את זמן הריצה של האלגוריתם.

הרעיון הוא שלא ניתן לעצים להיות יותר מדי גדולים. כלומר, לא נבצע פעולות `Link` אם אנחנו עומדים לחבר שני עצים שהגודל הכולל שלהם גדול מ־k[1].

מה יקרוס באנליזה המקורית שלנו? לא ניתן לחייב פעולות `TreepushRelabel` שלא מבצעות `Link`, `Cut` ולא מקדמות את הקשת הנוכחית.

איך נתקן? יש לכל היותר $\mathcal{O}(m \cdot n)$ Push־ים מרווים. לכן, נניח שכל הפעולות האלה מבצעות Push שאינו מרווה.

[1] נכון לעכשיו, k הוא פרמטר של האלגוריתם. בהמשך נמצא מה הערך המתאים לו ביותר.

נסמן ב־$(\star)$ את מספר ה־TreepushRelabel שלא מבצעים Link או Cut, וגם ה־Push הוא לא־מרווה (כלומר Push על (v,w) שהופך את v מפעיל ללא פעיל). נשאר לספור כמה צמתים הופכים להיות פעילים בפעולות TreepushRelabel בפעולות מהסוג $(\star)$. נחסום את מספר הפעולות מהסוג $(\star)$ על ידי חסימת מספר הפעמים שצומת הופך להיות פעיל.

מספר הצמתים שהופכים להיות פעילים ב־TreepushRelabel שאינם מהסוג $(\star)$ הוא פרופורציוני למספר פעולות ה־Link ו־Cut, כלומר $\mathcal{O}(m \cdot n)$ (כמו קודם). בכל פעולה מהסוג $(\star)$, ניתן להניח ש־v הופך ללא פעיל ו־w הופך להיות פעיל. מכיוון שלא ביצענו Link או Cut, אחד מהעצים T_v ו־T_w (של v ו־w בהתאמה) גדול מ־$k/2$. נחייב את הפעולה על העץ.

מכיוון שהשורש של העץ שאנחנו מחייבים הפך להיות פעיל או לא פעיל (אבל המצב שלו השתנה), כל עץ מחוייב לכל היותר פעמיים במעבר על התור. אם העץ לא היה קיים בתחילת המעבר, אז אפשר להעביר את החיוב על הפעולה לפעולת ה־Link או Cut שיצרה אותו לראשונה.

לכן, כל פעולת Link מחוייבת פעם אחת, וכל פעולת Cut מחוייבת פעמיים. לכן, יש $\mathcal{O}(m \cdot n)$ חיובים כאלה. בתחילת המעבר היו לנו $\mathcal{O}(n/k)$ עצים גדולים, וכל אחד מהם מחוייב פעם אחת, לכן יש $\mathcal{O}\left(n^3/k\right)$ חיובים כאלה.

לכן, הצמתים הופכים להיות פעילים לכל היותר $\mathcal{O}\left(m \cdot n + n^3/k\right)$ פעמים. זה חוסם את מספר הפעולות מהסוג $(\star)$, וגם את מספר הפעולות על העצים הדינאמיים.

עבור $k = n^2/m$, נקבל את החסם של $\mathcal{O}\left(m \cdot n \cdot \log n^2/m\right)$.

פרק 5

זרימה בעלות מינימלית

5.1 הגדרת הבעיה

נתון גרף מכוון $G = (V, E)$. נתונה פונקציית הקיבול מוגדרת לכל זוג צמתים $v, w \in V$. אם $(v, w) \notin E$, אזי $u(v, w) = 0$. כמו כן, נתונה גם פונקציית העלות מוגדרת לכל $(v, w) \in E$, ומסומנת ב־$c(v, w)$. נשים לב כי $c(v, w) = -c(w, v)$. כמו כן, בדומה למקרה של זרימה רגילה, נתונים שני צמתים s ו־t.

הגדרה 53 (זרימה): זרימה היא פונקציה על קשתות הגרף המקיימת:

$$f(v, w) = -f(w, v)$$

$$f(v, w) \leq u(v, w)$$

כמו כן, לכל צומת v חוץ מ־s ו־t מתקיים:

$$\sum_{w} f(v, w) = 0$$

נגדיר את הערך של הזרימה להיות:

$$|f| = \sum_{w} f(s, w)$$

נגדיר את העלות של הזרימה להיות:

$$\text{cost}(f) = \frac{1}{2} \cdot \sum_{(v,w) \in E} c(v, w) \cdot f(v, w)$$

בעיית הזרימה בעלות מינימלית (Minimum Cost Flow) עוסקת במציאת זרימה מקסימלית בעלת עלות מינימלית.

הגדרה 54 (מחזור): מחזור ($Circulation$) הוא פונקציה על קשתות הגרף המקיימת:

$$f(v,w) = -f(v,w)$$

$$f(v,w) \leq u(v,w)$$

כמו כן, לכל צומת v (כולל s ו־t):

$$\sum_{w} f(v,w) = 0$$

נגדיר את העלות של המחזור להיות:

$$\text{cost}(f) = \frac{1}{2} \cdot \sum_{(v,w)\in E} c(v,w) \cdot f(v,w)$$

בעיית המחזור בעלות מינימלית (Minimum Cost Circulation) עוסקת במציאת מחזור בעל עלות מינימלית.

משפט 55: הבעיות זרימה בעלות מינימלית ומחזור בעלות מינימלי הינן שקולות, דהיינו בהינתן אלגוריתם לפתרון אחת הבעיות, ניתן לפתור את הבעיה השניה.

הוכחה: נוכיח כל אחד מהכיוונים בנפרד:

מזרימה בעלות מינימלית למחזור בעלות מינימלית נניח שאנחנו יודעים למצוא זרימה בעלות מינימלית. יהי גרף מכוון $G = (V,E)$. נראה איך למצוא מחזור בעלות מינימלית על G.

נגדיר את הגרף $G' = (V',E)$ על ידי $V' = V \cup \{s,t\}$ (כאשר s ו־t הם שני צמתים חדשים בגרף). נמצא זרימה בעלת עלות מינימלית ב־G', ונקרא לה f.

נשים לב לכך שתחום ההגדרה של f הוא E, ולכן f גם מוגדרת היטב על G. לכן, f היא מחזור בעלות מינימלית על G.

ייתכנו מקרים בהם לקשתות בגרף יש עלות שלילית, ולכן הפתרון הטריוויאלי של זרימה שאינה מזרימה דבר אינו בהכרח הפתרון הנכון לזרימה בעלות מינימלית. זו גם הסיבה שמחזור בעלות מינימלית אינו בהכרח מחזור שאינו מזרים דבר.

ממחזור בעלות מינימלית לזרימה בעלות מינימלית נניח שאנחנו יודעים למצוא מחזור בעלות מינימלית. יהי גרף $G = (V,E)$, כך שהקיבולים בו חסומים על ידי U, והעלויות חסומות על ידי C. יהיו שני צמתים $s,t \in V$. נראה איך למצוא זרימה בעלות מינימלית על G.

נגדיר את הגרף $G' = (V,E')$ על ידי $E' = E \cup \{(t,s)\}$, ונגדיר $u(t,s) = m \cdot U$ (כאשר $|E| = m$) ו־$c(t,s) = -(C+1) \cdot n$ (כאשר $|V| = n$). נמצא מחזור בעלות מינימלית ב־G', ונקרא לו f.

אם נסתכל על f בלי הקשת (t,s), נקבל כי זוהי זרימה בעלות מינימלית. ■

הגדרה 56 (קיבול שיורי): הקיבול השיורי של זרימה f הוא פונקציה r על קשתות הגרף כך ש־$r(v,w) = u(v,w) - f(v,w)$.

כלומר, הקיבול השיורי הוא הערך אותו ניתן להזרים על (v,w) עד לרוויה של (v,w).

אלגוריתם 1.5 אלגוריתם ביטול המעגלים

```
repeat
    Find a negative cost residual cycle
    Saturate the cycle
until there are no negative cost residual cycles
```

5.2 אלגוריתם ביטול המעגלים

משפט 57 (קריטריון האופטימליות 1): f היא מחזור בעלות מינימלית אם ורק אם אין מעגלים בעלות שלילית ברשת השיורית.

הוכחה: ברור שאם f היא מחזור בעלות מינימלית אזי אין מעגלים בעלות שלילית ברשת השיורית (אחרת, היינו מוסיפים אותם ל־f, ומקטינים את העלות של f).
נניח ש־f היא לא אופטימלית (כלומר העלות אינה מינימלית), ויהי מחזור $f^\star$ כך ש־$\text{cost}(f^\star) < \text{cost}(f)$. אזי $f^\star - f$ היא מחזור על הרשת השיורית של f. כמו כן, $\text{cost}(f^\star - f) < 0$. לכן, $f^\star - f$ מכילה מעגל בעלות שלילית, בסתירה. ■

מהקריטריון הזה, נקבל את אלגוריתם ביטול המעגלים (Cycle Canceling Algorithm), כפי שמתואר באלגוריתם 1.5.
אם אנחנו מתחילים עם קיבולים שלמים, אזי גם הקיבולים השיוריים נשארים שלמים. בפרט, המחזור בעלות המינימלית הוא שלם.
איך אנחנו מוצאים מעגל שיורי בעלות מינימלית? אפשר לעשות את זה עם האלגוריתם של Bellman-Ford למסלולים קצרים ביותר בזמן $\mathcal{O}(m \cdot n)$ (לכל מעגל שלילי).

5.2.1 ניתוח

נניח שכל הקיבולים והעלויות של קשתות הגרף הם שלמים. נניח גם שלכל קשת $(v,w) \in E$, $u(v,w) \leq U$ ו־$|c(v,w)| \leq C$.
כמה איטרציות האלגוריתם יעשה? $m \cdot U \cdot C$, מכיוון שהעלות לא יכולה להיות קטנה מ־$-m \cdot U \cdot C$. מכיוון שבכל איטרציה משקיעים $\mathcal{O}(m \cdot n)$ עבודה (על Bellman-Ford), זמן הריצה הכולל של האלגוריתם הוא $\mathcal{O}\left(m^2 \cdot n \cdot U \cdot C\right)$.

5.3 פוטנציאל בצמתים

5.3.1 הגדרות

הגדרה 58 (פוטנציאל הצמתים): פונקציית פוטנציאל לצמתים היא פונקציה שממפה את הצמתים לממשיים, כלומר $\pi : V \to \mathbb{R}$.

באינטואיציה, באופן מקביל ל־Distance Labels שבהם השתמשנו בבעיית הזרימה המקסימלית, נרצה לקבל שעבור הקשת $(v,w) \in E$, $c(v,w) + \pi(w) - \pi(v) \geq 0$ עבור קשתות שיוריות.

הגדרה 59 (Reduced Costs): תהי קשת $(v,w) \in E$. נגדיר את ה־*Reduced Cost* שלה לפי פונקציית הפוטנציאל π על ידי:

$$c^\pi(v,w) = c(v,w) + \pi(w) - \pi(v)$$

טענה 60: נשים לב כי:

$$c^{\pi}(v,w) = -c^{\pi}(w,v)$$

הוכחה:

$$\begin{aligned} c^{\pi}(v,w) &= c(v,w) + \pi(w) - \pi(v) = -c(w,v) + \pi(w) - \pi(v) \\ &= -(c(w,v) + \pi(v) - \pi(w)) = -c^{\pi}(w,v) \end{aligned}$$

■

5.3.2 קריטריון האופטימליות

משפט 61 (קריטריון האופטימליות 2): f היא מחזור בעלות מינימלית אם ורק אם קיימת פונקציית פוטנציאל π כך ש־$c^{\pi}(v,w) \geq 0$ לכל קשת שיורית (v,w).

הוכחה: נניח ש־f היא מחזור בעלות מינימלית. נוכיח שקיימת פונקציית פוטנציאל π כך שלכל קשת שיורית (v,w), $c^{\pi}(v,w) \geq 0$.
נבחר צומת כלשהו t. יהי $\delta(v,t)$ המרחק הקצר ביותר מ־v ל־t ברשת השיורית של f. δ מוגדרת היטב, מכיוון שניתן להניח שהרשת השיורית קשירה היטב, ומקריטריון האופטימליות 1 (משפט 57), אין מעגלים שליליים ברשת השיורית. נגדיר:

$$\pi(v) = \delta(v,t)$$

תהי (v,w) קשת ברשת השיורית. אזי:

$$c^{\pi}(v,w) = c(v,w) + \pi(w) - \pi(v) = c(v,w) + \delta(w,t) - \delta(v,t)$$

אם המסלול הקצר ביותר מ־v ל־t עובר דרך w, אזי $\delta(v,t) = c(v,w) + \delta(w,t)$. אחרת, $\delta(v,t) < c(v,w) + \delta(w,t)$, ולכן בכל מקרה $c^{\pi}(v,w) \geq 0$.
נניח שקיימת פונקציית פוטנציאל π כך ש־$c^{\pi}(v,w) \geq 0$ לכל קשת שיורית (v,w). נוכיח ש־f היא מחזור בעלות מינימלית.
יהי $(v_1, v_2, \ldots, v_k)$ מעגל ברשת השיורית. אזי:

$$\begin{aligned} c^{\pi}(v_1,v_2) &= c(v_1,v_2) + \pi(v_2) - \pi(v_1) \\ c^{\pi}(v_2,v_3) &= c(v_2,v_3) + \pi(v_3) - \pi(v_2) \\ &\vdots \\ c^{\pi}(v_{k-1},v_k) &= c(v_{k-1},v_k) + \pi(v_k) - \pi(v_{k-1}) \\ c^{\pi}(v_k,v_1) &= c(v_k,v_1) + \pi(v_1) - \pi(v_k) \end{aligned}$$

נסכום את כל האיברים ונקבל את סך העלות של המעגל השיורי:

$$\begin{aligned} 0 \leq \sum_{i=1}^{k-1} c^{\pi}(v_i, v_{i+1}) + c^{\pi}(v_k, v_1) &= \sum_{i=1}^{k-1} [c(v_i, v_{i+1}) + \pi(v_{i+1}) - \pi(v_i)] \\ &\quad + c(v_k, v_1) + \pi(v_1) - \pi(v_k) \\ &= \cancel{\sum_{i=1}^{k} \pi(v_i)} - \cancel{\sum_{i=1}^{k} \pi(v_i)} \\ &\quad + \sum_{i=1}^{k-1} c(v_i, v_{i+1}) + c(v_k, v_1) \end{aligned}$$

לכן, אין מעגלים שליליים ברשת השיורית. מקריטריון האופטימליות 1 (משפט 57), נקבל כי f היא מחזור בעלות מינימלית. ■

5.3.3 האלגוריתם

הרעיון הוא כזה: מתחילים עם פוטנציאלים $\pi(v) = 0$ לכל $v \in V$. אחר כך מרווים כל קשת עם Reduced Cost שלילי. בשלב הזה קיבלנו פסאודו־זרימה, כי לחלק מהצמתים יש יותר זרימה נכנסת ויוצאת, וחלק מהצמתים הפוכים. השלב האחרון יהיה להעביר זרימה מצמתים עם עודף לצמתים עם מחסור.

את השלב האחרון מבצעים רק על קשתות עם $c^{\pi}(v, w) = 0$. אם אין קשתות כאלה, משנים את π כך שיהיו קשתות כאלה.

5.4 אופטימליות מקורבת

הגדרה 62 (ε־אופטמליות): נניח שקיימת פונקציית פוטנציאל π כך ש־$c^{\pi}(v, w) \geq -\varepsilon$ לכל קשת שיורית (v, w). אזי נאמר ש־f היא ε־אופטימלית.

ההגדרה תופסת לגבי זרימות, וגם לגבי פסאודו־זרימות.

למה 63: אם כל העלויות שלמות, אזי f היא ε־אופטימלית עבור $\varepsilon < 1/n$ אם ורק אם f היא אופטימלית (בעלות מינימלית).

הוכחה: אם f היא אופטימלית, ברור שהיא גם ε־אופטימלית.

אם f היא ε־אופטימלית, קיים פוטנציאל π כך שלכל מעגל C, ולכל קשת $(v, w) \in C$ במעגל, $c^{\pi}(v, w) > -1/n$. לכן:

$$c^{\pi}(C) = \sum_{(v,w) \in C} c^{\pi}(v, w) > n \cdot \left(-\frac{1}{n}\right) = -1$$

כמו כן, $c(C) = c^{\pi}(C) > -1$, ולכן $c(C) \geq 0$. לכן, f אופטימלית. ■

אלגוריתם 2.5 קירוב הרצף

```
1: ε ← C
2: for all v ∈ V do π(v) ← 0
3: for all (v, w) ∈ E do f(v, w) ← 0
4: while ε ≥ 1/n do
5:     ε ← ε/2
6:     Refine(f, π)
7: end while
8: return f

9: procedure Refine(f, π)
10:    for all (v, w) ∈ E do
11:    if c^π(v, w) < 0 then f(v, w) ← u(v, w)
12:    while there is an active vertex v do PushRelabel(v)
13: end procedure

14: procedure PushRelabel(v)
15:    if there is an admissible arc (v, w) then
16:        Push δ = min{e(v), r(v, w)} flow from v to w
17:    else
18:        Relabel(v)
19:    end if
20: end procedure

Require: v is active, ∀w.r(v, w) > 0 and c^π(v, w) ≥ 0
21: procedure Relabel(v)
22:    π(v) ← min_{r(v,w)>0} {π(w) + c(v, w) + ε}
23: end procedure
```

5.5 אלגוריתם קירוב הרצף

אלגוריתם קירוב הרצף (The Successive Approximation Algorithm) מתואר באלגוריתם 2.5.

במקרה הזה, נאמר שהקשת (v,w) היא *Admissible* אם $r(v,w)>0$ ו־$c^{\pi}(v,w)<0$.

נשים לב לכך שכל עוד יש צומת פעיל, אפשר להמשיך לבצע push או relabel. כמו כן, push ו־relabel משמרים את ה־ε־אופטימליות. לכן, כשהאלגוריתם מסיים, יש לנו זרימה שהיא ε־אופטימלית.

נשים לב ש־$\texttt{Relabel}(v)$ מגדיל את הפוטנציאל של v בלפחות ε. אבל הפוטנציאלים לא יכולים לגדול כל כך הרבה.

ניזכר מה קרה כשעסקנו בזרימה מקסימלית: אם יש לנו קבוצת צמתים Y, כך ש־$e(Y)\leq 0$, ו־$v\in Y$ עם $e(v)>0$, אזי חייבים להגיע לאיזשהו $w\in Y$ כך ש־$e(w)<0$. אפשר להוכיח את זה בקלות באינדוקציה.

בצורה יותר פורמלית, הוכחנו את הלמה הבאה:

למה 64: תהי f פסאודו־זרימה, ויהי f' מחזור. יהי v צומת פעיל ביחס ל־f. אזי קיים מסלול P מ־v לצומת w עם $e(w)<0$, כך שכל הקשתות על P הן שיוריות ביחס ל־f, וכל הקשתות על המסלול ההפוך ל־P הן שיוריות ביחס ל־f'.

5.5.1 אנליזה

ננסה לחסום את מספר הפעמים שמבצעים relabel.

למה 65: הפוטנציאל של צומת לא יכול לגדול ביותר מ־$3\cdot n\cdot\varepsilon$.

הוכחה: נסתכל על הגדילה ב־$\pi(v)$ עבור מסלול בגודל 4 – (v,v_1,v_2,v_3,w). אזי:

$$\begin{aligned}
\pi(v) &\leq \pi(v_1)+c(v,v_1)+\varepsilon\leq \pi(v_2)+c(v_1,v_2)+c(v,v_1)+2\cdot\varepsilon\leq\cdots\\
&\leq \pi(w)+c(v,v_1,v_2,v_3,w)+4\cdot\varepsilon=\pi'(w)+c(v,v_1,v_2,v_3,w)+4\cdot\varepsilon\\
&\leq \pi'(v_3)+c(w,v_3)+2\cdot\varepsilon+c(v,v_1,v_2,v_3,w)+4\cdot\varepsilon\leq\cdots\\
&\leq \pi'(v)+c(w,v_3,v_2,v_1,v)+8\cdot\varepsilon+c(v,v_1,v_2,v_3,w)+4\cdot\varepsilon\\
&= \pi'(v)+12\cdot\varepsilon
\end{aligned}$$

באופן כללי, אם אורך המסלול הוא ℓ, נקבל ש־$\pi(v)\leq\pi'(v)+3\cdot\ell\cdot\varepsilon\leq\pi'(v)+3\cdot n\cdot\varepsilon$. ■

מסקנה 66: יש לנו $\mathcal{O}(n^2)$ פעמים *relabel*.

ננסה עכשיו לחסום את מספר הפעמים שמבצעים push מרווה.

למה 67: יש $\mathcal{O}(m\cdot n)$ פעולות *push* מרווה.

הוכחה: מבצעים push רק כאשר $c^{\pi}(v,w)=c(v,w)+\pi(w)-\pi(v)<0$. לכן, כדי לבצע שוב פעולה כזו, צריך לדחוף זרימה בכיוון ההפוך, כדי ש־$\pi(w)$ יגדל ואז לחכות ש־$\pi(v)$ יגדל גם הוא. ■

נרצה עכשיו להראות שתת־הגרף המורכב מהקשתות שהן Admissible הוא חסר מעגלים.

למה 68: לאחר ביצוע *relabel* ל־v, אין קשתות *Admissible* שנכנסות ל־v.

הוכחה: לפני ש־$c^{\pi}(w,v) \geq -\varepsilon$, ה־relabel מגדיל את הערך של $\pi(v)$ בלפחות ε. ■

מסקנה 69: תת הגרף המורכב מקשתות שהן *Admissible* הוא חסר מעגלים.

הוכחה: באינדוקציה, כי פעולות push ו־relabel שומרות על התכונה הזאת. ■

עכשיו נרצה לטפל בפעולות Push שאינן מרוות.

למה 70: יש לכל היותר $\mathcal{O}(m \cdot n^2)$ פעולות *Push* שאינן מרוות.

הוכחה: נגדיר את פונקציית הפוטנציאל:

$$\Phi = \sum_{v \text{is active}} \Phi(v)$$

כאשר $\Phi(v)$ הוא מספר הצמתים הנגישים מ־v ברשת השיורית.
בפעולת Push שאינה מרווה מ־v ל־w, w עדיין נגיש מ־v. ממסקנה 69, נקבל שהרשת השיורית היא חסרת מעגלים, ולכן $\Phi(v) > \Phi(w)$ לכל w. מכיוון שלאחר ביצוע ה־Push, v הפך ללא פעיל (אחרת ה־Push היה מרווה), הפעולה הקטינה את Φ.

הערה 71: הקשת ההפוכה ((w,v)) עשוייה להפוך לקשת שיורית, אבל היא לא תהיה Admissible.

פעולת Push מ־v ל־w שמרווה את הקשת (v,w) עשוייה להגדיל את Φ בלכל היותר n אם w הפך להיות פעיל.
פעולת Relabel עשוייה להגדיל את $\Phi(v)$ בלכל היותר n, אבל מלמה 68, אין קשת שהיא Admissible שנכנסת ל־v, ולכן לכל $w \neq v$, $\Phi(w)$ לא יגדל. ■

5.5.2 מימוש גנרי ב־$\mathcal{O}(n^2 \cdot m)$

נתחזק את מצביע "הקשת הנוכחית" לכל צומת v. המצביע יוכל להצביע לכל קשת היוצאת מ־v, לא רק שיוריות או Admissible. את תיאור המימוש ניתן למצוא באלגוריתם 3.5.
באופן דומה למימוש המקביל בבעיית הזרימה המקסימלית, צריך להוכיח שמבצעים Relabel רק אם זה חוקי לבצע Relabel. זה ייתן לנו מימוש של $\mathcal{O}(n^2 \cdot m)$.

5.5.3 מימוש ב־$\mathcal{O}(n^3)$

בניגוד למה שראינו בזרימה מקסימלית, לא ידוע אם תור FIFO עוזר לנו כאן, כי האלגוריתם יכול לבצע $\mathcal{O}(n^3)$ מעברים על התור (ידועה דוגמה לכך), וכאן לא ספרנו את העבודה בתוך המעברים.
אנחנו נבצע `Discharge` לצמתים לפי סידור טופולוגי של תת־הגרף ה־Admissible. פעולת ה־`Discharge` מתוארת באלגוריתם 4.5. האלגוריתם הזה נקרא First Active Method.
וביתר פירוט: נתחזק רשימה של צמתים L, שתכיל מיון טופוגרפי של הצמתים ביחס לתת־הגרף ה־Admissible. נתחזק גם מצביע ל"צומת הנוכחי" ב־L. אם הוא פעיל, נבצע עליו `Discharge`. אחרת, נעבור לצומת הבא. אם היה relabel ל־v, נזיז אותו לראש הרשימה, והצומת הנוכחי נשאר v.

הערה 72: נשים לב ש־L הוא מיון טופולוגי של תת־הגרף ה־Admissible בכל רגע נתון באלגוריתם.

הערה 73: נשים לב שכשעושים $\texttt{Discharge}(v)$, v הוא הצומת הפעיל הראשון ב־L.

אלגוריתם 3.5 מימוש גנרי לאלגוריתם קירוב הרצף

```
1: procedure PushRelabel(v)
2:     Let (v, w) be the current edge of v
3:     if (v, w) is admissible then
4:         Push(v, w)
5:     else if (v, w) is not the last edge then
6:         Advance the current edge of v
7:         else
8:         Relabel(v)
9:         Set the current edge to the first edge
10:    end if
11: end procedure
```

אלגוריתם 4.5 פעולת ה־Discharge

```
1: procedure Discharge(v)
2:     repeat
3:         PushRelabel(v)
4:     until v is inactive or relabeled
5: end procedure
```

ניתוח סיבוכיות נגדיר מהלך באלגוריתם באופן הבא: הוא מתחיל כאשר הצומת הנוכחי הוא הצומת הראשון ב־L, ומסתיים כאשר מבצעים Relabel. ממסקנה 66, יש לכל היותר $\mathcal{O}\left(n^2\right)$ פעולות relabel, ולכן יש $\mathcal{O}\left(n^2\right)$ מעברים.

כמו כן, נשים לב שבכל מעבר באלגוריתם, יש לכל היותר פעולת Push אחת שאינה מרווה לכל צומת. לכן, כל מעבר לוקח $\mathcal{O}\left(n\right)$ זמן.

לכן, בסך הכל, זמן הריצה של האלגוריתם הוא $\mathcal{O}\left(n^3\right)$.

הערה 74: מציאה של צומת פעיל ב־L לא לוקחת זמן קבוע כפי שקל לחשוב. בכל מעבר, מושקעת $\mathcal{O}\left(n\right)$ עבודה במציאת צומת פעיל. מכיוון שיש $\mathcal{O}\left(n^2\right)$ מהלכים, נשקיע בריצת האלגוריתם $\mathcal{O}\left(n^3\right)$ זמן במציאת צומת פעיל ב־L. למזלנו, זה לא משנה את הסיבוכיות הכוללת של האלגוריתם.

5.6 שימוש בעצים דינאמיים

נתחזק יער דינאמי של חלק מהקשתות הנוכחיות שהן Admissible. הצמתים הפעילים יהיו בשורשים של העצים. זה יהיה בדיוק כמו במקרה של זרימה מקסימלית, פרט למימוש של TreepushRelabel, שמתואר באלגוריתם 5.5.

נשים לב לעובדות הבאות:

הערה 75: היער הדינאמי נשאר יער כי תת־הגרף ה־Admissible הוא חסר מעגלים.

למה 76: אחרי ביצוע *TreepushRelabel*, ה־*excess* נשאר רק בשורשים.

אלגוריתם 5.5 TreepushRelabel (v)

```
1: if the current edge (v, w) is admissible then
2:     Push from v to w a flow of min {e (v) , r_f (v, w)}
3:     if (v, w) is not saturated and SIZE(T_v) + SIZE(T_w) < k then
       LINK(v, w, r_f (v, w))
4:     repeat
5:         w, c ← FINDMIN(w)
6:         c ← min {c, e (w)}
7:         ADDCOST(w, −c)
8:         repeat
9:             u, c ← FINDMIN(w)
10:            if c = 0 then CUT(u)
11:        until c ≠ 0
12:    until e (w) = 0 or w is not a root
13: else                                   ▷ (v, w) is not admissible
14:    if (v, w) is not the last edge then
15:        Advance the current edge
16:    else
17:        RELABEL(v)
18:        for all child u of v do CUT(u)
19:    end if
20: end if
```

כמו כן:

טענה 77: הסדר של L משתנה כשעושים Relabel, וכל צומת מבצע Discharge פעם אחת בין שתי פעולות Relabel.

טענה 78: הגודל של כל עץ הדינאמי הוא לכל היותר k.

למה 79: מספר הפעולות של עצים בינאריים (וגם פעולות אחרת) שמבצעים הוא $\mathcal{O}(m \cdot n)$ ועוד מספר הפעמים שצומת הופך לפעיל.

הוכחה: אפשר לחייב כל פעולה ל־Push מרווה או Cut. בכל מקרה אחר, שורש הופך ללא פעיל. ■

למה 80: מספר הפעמים שצומת הופך לפעיל הוא $\mathcal{O}\left(m \cdot n + {}^{n^3}/_{k}\right)$.

הוכחה: זה קורה בכל פעם שעוברים על קשת שהיא Admissible. בכל פעם כזאת, מספר הצמתים שהופכים לפעילים הוא 1 ועוד מספר הפעמים שמבצעים Cut.

בחלק מהמקרים, אפשר "לחייב" את הפעולה על פעולת Link, Cut או Push שאינו מרווה ($\mathcal{O}(m \cdot n)$). אלה לא המקרים המעניינים, כי הם כבר נספרו. מעניינים אותנו המקרים שבהם אי אפשר לחייב אותם על פעולה אחרת.

זה קורה כאשר לא מבצעים Link, כי החיבור של שני העצים יתן עץ גדול מדי. במקרה הזה, או שב־T_v יש יותר מ־${}^{k}/_{2}$ צמתים, או שב־T_w יש יותר מ־${}^{k}/_{2}$ צמתים. במקרה הזה, r (שהוא השורש של T_w) נהיה פעיל ו־v נהיה לא פעיל.

נחייב את ההפיכה של r לפעיל לעץ הגדול, אם הוא היה קיים בתחילת הפאזה. אחרת, נחייב אותו ל־Link או Cut שיצר את העץ. לכן, כל Link מחוייב פעם אחת וכל Cut מחוייב פעמיים. לכן, יש לנו $\mathcal{O}(m \cdot n)$ חיובים כאלה בכל הפאזות.

בתחילת הפאזה יש לנו $\mathcal{O}\left({}^{n}/_{k}\right)$ עצים גדולים (מ־${}^{k}/_{2}$ צמתים), וכל אחד מהם מחוייב פעם אחת. בסך הכל, $\mathcal{O}\left({}^{n^3}/_{k}\right)$. ■

לכן, בסך הכל, קיבלנו $\mathcal{O}\left(m \cdot n + {}^{n^3}/_{k}\right)$ פעולות של עצים דינאמיים, והעלות של כל אחת מהן היא $\mathcal{O}(\log k)$. בסך הכל:

$$\mathcal{O}\left(m \cdot n \cdot \log k + \frac{n^3}{k} \cdot \log k\right)$$

עבור $k = \frac{n^2}{m}$, נקבל:

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

אבל עדיין לא סיימנו. עדיין יש סיכוי שאנחנו עוברים על כל L בין כל שתי פעולות Relabel. צריך למצוא צומת פעיל בצורה יעילה יותר.

5.6.1 ייצוג טוב יותר של L

במקום לייצג את L בתור רשימה של צמתים, נגיד ש־L היא רשימה של רשימות של צמתים. בכל פעם "שוברים" רשימה (כלומר עוברים לרשימה חדשה) כאשר מגיעים לצומת פעיל.

באופן הזה, אנחנו מקבלים את התכונות הבאות בפשטות:

- גישה לצומת הפעיל הראשון (בראש הרשימה הראשונה או השנייה).
- כשצומת הופך ללא־פעיל, מאחדים שתי רשימות לרשימה אחת (באמצעות שרשור).
- כשמבצעים Relabel לצומת (פעיל) v, מוחקים אותו מהרשימה שלו, מאחדים את הרשימה שלו עם הרשימה הקודמת, ומעבירים אותו לתחילת הרשימה הראשונה.
- כשצומת הופך להיות פעיל, צריך לפצל את הרשימה שבה הוא נמצא, כך שהוא יפתח רשימה חדשה.

למעשה, אנחנו רוצים מבנה נתונים שיספק לנו את הפעולות הבאות (בשביל הרשימות הפנימיות):

- הכנסה והוצאה בתחילת הרשימה.

  נעשה את זה בזמן $\mathcal{O}(1)$.
- שרשור.

נניח שמשרשרים שתי רשימות בגדלים ℓ_1 ו־ℓ_2. נעשה את השרשור בזמן $\mathcal{O}(\log\min\{\ell_1,\ell_2\})$.

- פיצול בנקודה כלשהי.

  נעשה את זה בזמן $\mathcal{O}(1)$.

בהינתן הזמנים האלה, נצליח לבצע הכנסות, הוצאות ושרשורים בזמן ביצוע Relabel בזמן כולל של $\mathcal{O}\left(n^2\cdot\log n\right)$. כשצומת יהפוך לפעיל, הפיצול יקח $\mathcal{O}\left(m\cdot n+{}^{n^3}/_{k}\right)$. מה קורה עם שרשור בין שתי פעולות Relabel?

יש לכל היותר $\mathcal{O}\left(m\cdot n+{}^{n^3}/_{k}\right)$ שרשורים כאלה. נפריד ל־2 מקרים: שרשורים עם רשימות קטנות (עם לכל היותר k צמתים), ושרשורים עם רשימות גדולות (עם יותר מ־k צמתים).

יש לכל היותר $\mathcal{O}\left(m\cdot n+{}^{n^3}/_{k}\right)$ שרשורים קטנים. כל אחד מהם לוקח $\mathcal{O}(\log k)$ זמן, ולכן בסך הכל, מושקעת $\mathcal{O}\left(\left(m\cdot n+{}^{n^3}/_{k}\right)\cdot\log k\right)$ עבודה בשרשורים קטנים.

נבחן כעת שרשורים של רשימות גדולות: נתבונן בסדרת השרשורים במעבר בודד (בין שתי פעולות Relabel). נסמן ב־x_i את גודל הרשימה ה־i־ית מייד בתחילת המעבר. $\sum_i x_i = n$, ולכן מספר ה־x_i־ים הגדולים מ־k הוא לכל היותר ${}^{n}/_{k}$. $\sum_i \log x_i$ הוא חסם על עלות כל השרשורים. נזרוק מהסכום את כמות השרשורים הקטנים שכבר ספרנו. $\sum_{i|x_i>k}\log x_i$ הוא חסם על עלות השרשורים הגזוליס במהלך המעבר.

הסכום האחרון הוא סכום על לכל היותר ${}^{n}/_{k}$ מחוברים. לכן, קל להוכיח כי:

$$\sum_{i|x_i>k}\log x_i \le \frac{n}{k}\cdot\log k$$

כי הסכום הוא סכום של לכל היותר $\log {}^{n}/_{k}$־ים, שהארגומנטים שלהם מסתכמים לערך שהוא לכל היותר k.

לכן, הזמן הכולל לביצוע שרשורים של רשימות (קטנות וגדולות) הוא $\mathcal{O}\left(\left(m\cdot n+{}^{n^3}/_{k}\right)\cdot\log k\right)$.

עכשיו רק נותר לחשוב על ייצוג טוב למבנה הנתונים שיספק את הביצועים הנדרשים.

אפשר לחשוב על ייצוג פשוט באמצעות רשימה מקושרת דו כיוונית. הבעייה בייצוג הפשוט של רשימה מקושרת דו־כיוונית היא מציאת המקום ברשימת הרשימות שאליו צריך להכניס את הרשימה החדשה שהתקבלה מהפיצול. לא נוכל לעשות את זה בזמן קבוע.

אבל Finger Search Trees יכולים לעזור לנו כאן. הרעיון הוא פשוט: לוקחים עץ חיפוש בינארי מאוזן רגיל (שבו כל צומת מצביע לאבא שלו), אבל מתחילים את החיפוש מהבן הכי ימני או שמאלי, ולא מהשורש. עץ כזה עושה חיפוש בזמן $\mathcal{O}(\log d)$, כאשר d הוא המרחק (באלמנטים) לקצה הקרוב לנקודה שאותה מחפשים. זה נכון גם להכנסת איברים (ב־Amortized) ולמחיקת איברים (במקרה הגרוע).

ביתר דיוק, מה שמעניין אותנו: זמן Amortized לפיצול של עץ הוא $\mathcal{O}(1)$. זמן Amortized לשרשור של שני עצים בגודל ℓ_1 ו־ℓ_2 הוא $\mathcal{O}(\log\min\{\ell_1,\ell_2\})$. הזמן הדרוש להכנסת או הוצאת איבר מתחילת הרשימה הוא $\mathcal{O}(\log 1)=\mathcal{O}(1)$.

נסביר על זמן ה־Amortized של פיצול: הזמן בפועל הוא $\log\ell_1$. הפוטנציאל לפני הפעולה היה $-\log(\ell_1+\ell_2)$, ואחרי הפעולה הוא $-\log\ell_1-\log\ell_2$. כמו כן, נזכור כי $\ell_1\le\ell_2$. לכן, בסך הכל, זמן Amortized הוא:

$$\begin{aligned}\text{Amortized}(\texttt{Split}) &= \underbrace{\cancel{\log\ell_1}}_{\text{Actual}}+\underbrace{\log(\ell_1+\ell_2)-\cancel{\log\ell_1}-\log\ell_2}_{\Delta\Phi}\\ &= \log\frac{\ell_1+\ell_2}{\ell_2}=\log\left(\frac{\ell_1}{\ell_2}+1\right)\le\log 2=1=\mathcal{O}(1)\end{aligned}$$

5.7 גרסה פולינומית חזקה לאלגוריתם קירוב הרצף

5.7.1 פוטנציאלים ε־צמודים

בהינתן פסאודו־זרימה f, נרצה למצוא את ה־ε הקטן ביותר כך ש־f היא ε־אופטימלית.

משפט 81: f היא ε־אופטימלית אם ורק אם ממוצע העלות המינימלי של מעגל שיורי הוא לפחות $-\varepsilon$.

הוכחה: נניח ש־f היא ε־אופטימלית. יהי Γ מעגל שיורי בגודל ℓ לפי f. מההנחה, לכל $(v,w)\in\Gamma$, $c^\pi(v,w)\ge-\varepsilon$. לכן:

$$\frac{\sum_{(v,w)\in\Gamma}c(v,w)}{\ell}=\frac{\sum_{(v,w)\in\Gamma}c^\pi(v,w)}{\ell}\ge\frac{-\varepsilon\cdot\cancel{\ell}}{\cancel{\ell}}=-\varepsilon$$

לכן, ממוצע העלות של מעגל שיורי Γ הוא לפחות $-\varepsilon$, ולכן ממוצע העלות המינימלי של מעגל שיורי הוא לפחות $-\varepsilon$.

נניח כעת שממוצע העלות המינימלי של מעגל שיורי הוא לפחות $-\varepsilon$. נגדיר: $c^\varepsilon(v,w)=c(v,w)+\varepsilon$. אזי אין מעגל שיורי שלילי ביחס ל־c^ε.

נייצר עץ מסלולים קצרים מצומת כלשהו ברשת השיורית ביחס ל־c^ε. נקבל:

$$c^\varepsilon(v,w)+d(w)-d(v)\ge 0$$

לכן:

$$c(v,w)+d(w)-d(v)\ge-\varepsilon$$

נגדיר: $\pi(v)=d(v)$, וקיבלנו את המבוקש. ■

אלגוריתם 6.5 גרסה פולינומית חזקה לאלגוריתם קירוב הרצף

```
1: ε ← C
2: for all v ∈ V do π(v) ← 0
3: for all (v,w) ∈ E do f(v,w) ← 0
4: loop
5:     find η and π_η such that f is η-tight with respect to π_η
6:     if η = 0 then return
7:     ε ← η/2
8:     REFINE(f, π_η)
9: end loop
```

הגדרה 82 (η־צמידות): זרימה f היא η־צמודה (η-*tight*) אם לכל $\varepsilon < \eta$, f היא לא ε־אופטימלית, אבל f היא η־אופטימלית.

מסקנה 83: f היא ε־צמודה אם ורק אם ממוצע העלות המינימלי של מעגל שיורי הוא בדיוק $-\varepsilon$.

5.7.2 האלגוריתם

אנחנו יכולים למצוא את הערך של ממוצע העלות המינימלי של מעגל שיורי ב־$\mathcal{O}(m \cdot n)$ זמן (בעזרת האלגוריתם של Karp). אלגוריתם 6.5 מתאר את הגרסה הפולינומית החזקה של אלגוריתם קירוב הרצף.

החסם החדש על זמן הריצה של האלגוריתם (כפי שנוכיח בהמשך) יהיה:

$$\mathcal{O}\left(m \cdot n \cdot \log \frac{n^2}{m} \cdot \min\left\{\log\left(n \cdot C\right), m \cdot \log n\right\}\right) = \mathcal{O}\left(m^2 \cdot n \cdot \log \frac{n^2}{m} \cdot \log n\right)$$

זה פולינומי חזק, כי עכשיו החסם של זמן הריצה לא תלוי ב־C, שהוא החסם לעלויות של הקשתות.

5.7.3 אנליזה

משפט 84: נניח כי f היא ε־אופטימלית ביחס ל־π. כמו כן, נניח שקיימת קשת (v,c) כך ש־$|c^\pi(v,w)| \geq 2 \cdot n \cdot \varepsilon$. אזי בכל f' שהיא גם ε־אופטימלית, $f(v,w) = f'(v,w)$.

הוכחה: נניח בה"כ כי $c^\pi(v,w) \leq -2 \cdot n \cdot \varepsilon$ (אחרת $c^\pi(w,v) \leq -2 \cdot n \cdot \varepsilon$, וזה שקול). מכיוון ש־$f$ היא ε־אופטימלית, $f(v,w) = u(v,w)$, כי אין קשת שלילית ברשת השיורית.

נניח בשלילה ש־f' היא גם ε־אופטימלית, ו־$f'(v,w) \neq f(v,w)$. מכיוון ש־f מרווה את (v,w), $f'(v,w) < f(v,w)$, ולכן $f'(w,v) > f(w,v)$.

נסתכל על $f - f'$. היא גם מחזור (circulation), והיא חוקית (כלומר פיזיבילית) ברשת השיורית של f, כי לכל קשת (x,y):

$$f'(x,y) - f(x,y) \leq u(x,y) - f(x,y)$$

וזה הקיבול השיורי של הקשת (x,y). אזי $f - f'$ מכילה מעגל דרך (w,v), כי כל מחזור בנוי ממעגלים, ועל (w,v) יש זרימה חיובית. נקרא למעגל הזה Γ.

מכיוון ש־f היא ε־אופטימלית, $c^{\pi}(a,b) \geq -\varepsilon$ לכל קשת (a,b) במעגל Γ. נשים לב ש־Γ נמצא ברשת השיורית של f. לכן, העלות הכוללת של המעגל Γ, שהוא שיורי ביחס ל־f היא:

$$c(\Gamma) \geq 2 \cdot n \cdot \varepsilon + (n-1) \cdot (-\varepsilon) > n \cdot \varepsilon$$

המעגל בכיוון ההפוך שיורי ביחס ל־f', ועלותו היא לכל היותר $-n \cdot \varepsilon$. זאת סתירה, כי f' היא ε־אופטימלית. ■

הערה 85: נשים לב שגם $|C^{\pi}(w,v)| \geq 2 \cdot n \cdot \varepsilon$, ולכן גם $f(w,v) = f'(w,v)$.

מסקנה 86: תחת ההנחה במשפט *84*, כל f' אופטימלית מקיימת $f(v,w) = f'(v,w)$.

למה 87: נסמן ב־F_{ε} את קבוצת הקשתות שהזרימה בהן זהה לכל f שהיא ε־אופטימלית. נניח שקיימת f שהיא ε־צמודה. נסמן: $\varepsilon' = \frac{\varepsilon}{2 \cdot n}$.
אזי $F_{\varepsilon'}$ חלקי ממש ל־F_{ε}, כלומר $F_{\varepsilon'} \subsetneq F_{\varepsilon}$.

הוכחה: ברור ש־F_{ε} מכילה את $F_{\varepsilon'}$, מכיוון שזרימה שהיא ε'־אופטימלית היא גם ε־אופטימלית.
תהי π פונקציית הפוטנציאל כך שקיים מעגל שיורי. נקרא לו Γ. אזי לכל קשת (v,w) של Γ, $c^{\pi}(v,w) = -\varepsilon$. לכן, הקשתות של Γ לא נמצאות ב־F_{ε}, כי גם הזרימה ההפוכה של f על Γ תגרום לכל הקשתות להיות בעלות פוטנציאל של ε, וזה מותר.
נסתכל על f' ועל π' שמוכיח ש־f' היא ε'־אופטימלית. אזי קיימת קשת $(v,w) \in \Gamma$ כך ש־$c^{\pi'}(v,w) \leq -\varepsilon \leq -2 \cdot n \cdot \varepsilon'$. לכן, $(v,w) \in F_{\varepsilon'} \setminus F_{\varepsilon}$, כי העלות ביחס ל־$\pi'$ עשוייה להשתנות, אבל הממוצע צריך להישאר אותו ממוצע, כי סכום העלויות במעגל לא תלוי בפוטנציאלים. הפוטנציאלים רק משנים את הפיזור על הקשתות. ■

מסקנה 88: בכל לכל היותר $\log(2 \cdot n)$ צעדים, נאסוף לפחות עוד קשת אחת. מכאן נוסף החסם של $\mathcal{O}(m \cdot \log n)$ על מספר האיטרציות.
לכן זמן הריצה הכולל של האלגוריתם הוא:

$$\mathcal{O}\left(n \cdot m \cdot \log \frac{n^2}{m} \cdot \min\left\{\log(n \cdot C), m \cdot \log n\right\}\right)$$

פרק 6

האלגוריתם לזרימה מקסימלית של Goldberd & Rao

6.1 הקדמה

נחזור לאלגוריתם של Dinic: זרימה היא חוסמת אם כל מסלול מ־s ל־t מכיל קשת רוויה. האלגוריתם עושה BFS מ־s, ומסתכל רק על תת־גרף של הרשת השיורית, שמושרה מה־BFS, ומוסיף קשתות.

נסמן ב־$d(v)$ את המרחק של הצומת v מ־t ברשת השיורית. Dinic מסתכל על תת־הגרף L של הרשת השיורית, שבו הקשת (v,w) קיימת אם ורק אם $d(w) = d(v) + 1$.

משפט 89: *Dinic* עושה לכל היותר $n-1$ חישובי זרימה חוסמת. זה נובע מכך ש־$d(s)$ קטן בלפחות 1 בכל חישוב של זרימה חוסמת.

הסיבוכיות הכוללת של האלגוריתם של Dinic היא $\mathcal{O}(n^2 \cdot m)$ בעזרת DFS פשוט. אם משתמשים בעצים דינאמיים, אפשר לרדת ל־$\mathcal{O}(m \cdot n \cdot \log n)$.

מה יקרה אם כל הקיבולים של הקשתות יהיו 0 או 1 בלבד? במקרה הזה, יותר קל למצוא זרימה חוסמת, כי אם מגיעים ל־s מ־t ברשת השיורית, בכל פעם מורידים את כל הקשתות מהמסלול, ואז אפשר למצוא זרימה חוסמת ב־$\mathcal{O}(m)$. זה מביא את האלגוריתם לזמן כולל של $\mathcal{O}(m \cdot n)$.

עדיין אפשר יותר טוב. נסתכל רק על מסלולים מ־t ל־s שארוכים מ־$\sqrt{m}$. אזי נסיים ב־$\sqrt{m}$ איטרציות. לכן, הזמן הוא $\mathcal{O}(m \cdot \sqrt{m}) = \mathcal{O}(m^{3/2})$. אם הגרף דליל, ירדנו מ־$\mathcal{O}(n^2)$ בזמן הריצה שלנו.

אפשר להסתכל על זה גם מהצד השני, ולומר שמבצעים לכל היותר $n^{2/3}$ חישובי זרימה חוסמת, ואז הזמן הכולל של האלגוריתם הוא $\mathcal{O}\left(\min\left\{n^{2/3}, \sqrt{m}\right\} \cdot m\right)$.

מה יקרה אם כל הקיבולים הם שלמים ב־$\{1, 2, \ldots, U\}$?

6.2 האלגוריתם של Goldberg & Rao

נתחזק חסם עליו F שמתאר את המרחק מהזרימה הנוכחית f לזרימה האופטימלית $f^\star$. בכל פאזה, נקטין את F ל־$F/2$. נרצה לעשות את זה בזמן $\mathcal{O}(m \cdot \sqrt{m})$. אם נצליח, נקבל אלגוריתם שרץ בזמן של $\mathcal{O}(m \cdot \sqrt{m} \cdot \log(n \cdot U))$.

למה $\log(n \cdot U)$ נכנס לסיבוכיות? כשמתחילים מזרימת ה־0, $F = n \cdot U$. אם כל פעם מחלקים פי 2, יש $\mathcal{O}(\log(n \cdot U))$ איטרציות.

בכל פאזה, נגדיר: $\Delta = \frac{F}{\sqrt{m}}$. ננסה לדחוף Δ זרימה בזמן $\mathcal{O}(m)$. אם נצליח, ברור שאנחנו עושים $\mathcal{O}(m \cdot \sqrt{m})$ עבודה בפאזה. לרוע המזל, לא תמיד נצליח, כי לא תמיד נצליח לדחוף Δ זרימה.

אם לא נצליח, יהיו לנו שני סוגים של קשתות: קשתות עם קיבול גדול (יותר גדול מ־$3 \cdot \Delta$), וקשתות עם קיבול קטן (כל השאר). נגדיר אורך של קשת צרה (קיבול קטן) להיות 1 ואורך של קשת רחבה (קיבול גדול) להיות 0 כשמודדים מרחקים מ־s (או t) ב־BFS. נקבל חתך של קשתות שחורות עם $2 \cdot \sqrt{m}$ קשתות, כשהקיבול של כל אחת הוא לכל היותר Δ. למעשה, הגדרנו כאן פונקציית מרחק בינארית ℓ.

לכל v נחשב את $d_\ell(v)$ – המרחק של v מ־t לפי פונקציית המרחק ℓ ברשת השיורית.

הגדרה 90 (קשת Admissible): נאמר כי הקשת השיורית (v,w) היא *Admissible* אם $d_\ell(v) = d_\ell(w) + \ell(v,w)$.

נגדיר את L להיות תת־הגרף של הרשת השיורית המורכב מהקשתות ה־Admissible. נחשב זרימה בערך Δ או זרימה חוסמת בתת־הגרף ה־Admissible. נראה בהמשך שאפשר לעשות את זה בזמן $\mathcal{O}(m)$. נזכור גם כי כל פעם אחרי שמחשבים זרימה כזו, הרשת השיורית משתנה, כמו גם פונקציית המרחק, וכו', וצריך לחשב הכל מחדש.

6.3 ניתוח

למה 91: נניח שהוספנו זרימה חוסמת בתת־הגרף ה־*Admissible*. יהיו ℓ ו־d_ℓ האורכים והמרחקים לפני העדכון, ו־ℓ' ו־$d_{\ell'}$ האורכים והמרחקים אחרי העדכון.
אזי, לכל קשת שיורית (v,w):

$$d_\ell(v) \le d_\ell(w) + \ell'(v,w)$$

הוכחה: עבור קשת שיורית (v,w) שהיא קשת גדולה (לפני העדכון), כלומר הקיבול שלה גדול מ־$3 \cdot \Delta$, $\ell(v,w) = 0$, ולכן:

$$d_\ell(v) \le d_\ell(w) + \ell(v,w) = d_\ell(w) \le d_\ell(w) + \ell'(v,w)$$

כי $\ell'(v,w) \ge 0$.

עבור קשת שיורית (v,w) שהיא קשת קטנה (לפני העדכון), כלומר הקיבול שלה הוא לכל היותר $3 \cdot \Delta$, $\ell(v,w) = 1$. כדי ש־$\ell'(v,w)$ יהיה 0, הקיבול השיורי של (v,w) צריך לגדול, אבל זה לא אפשרי, כי (w,v) היא לא Admissible.
נסביר: מכיוון ש־$\ell(v,w) = 1$,

$$d_\ell(v) \le d_\ell(w) + \ell(v,w) \lneq d_\ell(w)$$

לכן, לא ייתכן ש־$d_\ell(w) = d_\ell(v) + \ell(w,v)$, כי $d_\ell(w) < d_\ell(v)$ ו־$\ell(w,v) \ge 0$. לכן, (w,v) היא לא Admissible, ולכן הקיבול השיורי על (v,w) לא יכול לגדול, ולכן $\ell'(v,w) = 1 = \ell(v,w)$. לכן:

$$d_\ell(v) \le d_\ell(w) + \ell(v,w) = d_\ell(w) + 1 = d_\ell(w) + \ell'(v,w)$$

■

מסקנה 92: יהי מסלול שיורי $s, v_1, v_2, \ldots, t$. אזי:

$$\begin{aligned} d_\ell(s) &\leq d_\ell(v_1) + \ell'(s, v_1) \leq d_\ell(v_2) + \ell'(s, v_1) + \ell'(v_1, v_2) \leq \cdots \\ &\leq d_\ell(t) + d_{\ell'}(s) \end{aligned}$$

לכן, המרחקים לא קטנים אף פעם.

למה 93: נניח שהוספנו זרימה חוסמת בתת־הגרף ה־*Admissible*. יהיו ℓ ו־d_ℓ האורכים והמרחקים לפני העדכון, ו־ℓ' ו־$d_{\ell'}$ האורכים והמרחקים אחרי העדכון.
אזי קיימת רשת שיורית (v, w) על המסלול הקצר ביותר (לפי ℓ') מ־s ל־t כך ש:

$$d_\ell(v) < d_\ell(w) + \ell'(v, w)$$

הוכחה: מכיוון שהזרימה חוסמת, לפחות קשת אחת על המסלול הזה היא לא Admissible (לפי ℓ ו־d_ℓ). נסמן אותה ב־(v, w). אזי:

$$d_\ell(v) < d_\ell(w) + \ell(v, w)$$

במקרה הזה, הקשת יכולה ללכת אחורה ברמות (כלומר $d_\ell(v) < d_\ell(w)$), או ש־$\ell(v, w) = 1$ ו־$d_\ell(v) = d_\ell(w)$ (כלומר הקשת באותה הרמה).
אם $d_\ell(v) < d_\ell(w)$, אזי מכך ש־$\ell'(v, w) \geq 0$ נקבל:

$$d_\ell(v) < d_\ell(w) \leq d_\ell(w) + \ell'(v, w)$$

אם $d_\ell(v) = d_\ell(w)$ ו־$\ell(v, w) = 1$: אם $\ell'(v, w) = 1$, אז אנחנו עדיין בסדר. אם $\ell'(v, w) = 0$, יש לנו בעיה. במקרה הזה, הקיבול השיורי של (v, w) לפני העדכון היה קטן מ־$3 \cdot \Delta$, ואחרי העדכון הוא גדול מ־$3 \cdot \Delta$. זה אומר שהעדכון מוסיף זרימה על (w, v). לכן, (w, v) היא Admissible.
כרגע, שום דבר לא מונע מהמצב הזה לקרות. נקרא לקשתות מהסוג הזה פיוחדות. לכן, נתקן את ההגדרה של פונקציית המרחק ℓ באופן הבא: נגיד ש־$\ell(v, w) = 0$ אם $d_\ell(v) = d_\ell(w)$, $\ell(w, v) = 0$ ו־$u(v, w) \geq 2 \cdot \Delta$.
השינוי הזה לא פוגע בנכונות של למה 91, אבל מסיים את ההוכחה של הלמה הנוכחית.■

מסקנה 94: נסמן את המסלול השיורי הקצר ביותר מ־s ל־t (לפי ℓ') ב־$s, v_1, v_2, \ldots, t$. אזי:

$$\begin{aligned} d_\ell(s) &\leq d_\ell(v_1) + \ell'(s, v_1) \leq d_\ell(v_2) + \ell'(s, v_1) + \ell'(v_1, v_2) \leq \cdots \\ &< d_\ell(t) + d_{\ell'}(s) \end{aligned}$$

כלומר, המרחק מ־s ל־t גדל לאחר הוספת זרימה חוסמת.

אלגוריתם 1.6 Goldberg & Rao

```
1: F ← n · U
2: repeat
3:     c ← the smallest canonical cut
4:     if c < F/2 then F ← F/2
5:     Compute ℓ, modified on the special arcs, and d_ℓ
6:     Contract strongly connected components of 0-length arcs
7:     Compute a blocking or Δ-valued flow in the contracted admissible
   (acyclic) graph
8:     Add this flow to the current flow
9: until F < 1
```

6.4 המשך המימוש

איך אפשר למצוא זרימה עם ערך Δ או זרימה חוסמת? תת־הגרף ה־Admissible שלנו מכיל מעגלים (של קשתות במרחק 0).

נוכל להפוך אותו לגרף חסר מעגלים בעזרת איחוד של רכיבי קשירות חזקים (הנמצאים באותה הרמה, ומורכבים מקשתות שהמרחק שלהן הוא 0) לצומת בודד. זה לוקח $\mathcal{O}(m)$ זמן.

אם מצאנו זרימה שהערך שלה גדול מ־Δ, אז נצטרך להחזיר זרימה ל־s. נעשה את זה באמצעות מיון טופולוגי של הצמתים (כאשר חלק מהצמתים הם איחוד של רכיבי קשירות חזקים), ונחזיר את הזרימה לפי הסידור הזה.

איך ננתב את הזרימה בתוך רכיבי הקשירות החזקה שאיחדנו? נבחר צומת כלשהו z ברכיב. אנחנו יודעים שהקיבול השיורי של כל קשת ברכיב הוא לפחות $\Delta \cdot 2$, אבל דרך הרכיב הזרמנו לא יותר מ־Δ זרימה. נמצא עץ פורש הנכנס ל־z, ועץ פורש היוצא מ־z. נשלח את כל הזרימה הנכנסת ל־z בעזרת העץ הנכנס אליו, ואת כל הזרימה היוצאת מ־z בעזרת העץ היוצא ממנו.

כדי לשמור על החסם שלנו F מהזרימה האופטימלית, נחשב את הקיבול השיורי של כל חתך קנוני (S_k, T_k). אם הקיבול השיורי המינימלי מהצורה הזאת קטן מ־$F/2$, נעדכן את F ונתחיל את הפאזה הבאה.

6.5 סיכום

אלגוריתם 1.6 מתאר את האלגוריתם של Goldberg & Rao. זמן הריצה שקיבלנו הוא $\mathcal{O}(m \cdot \sqrt{m} \cdot \log n \cdot \log(n \cdot U))$. למעשה, עם אנליזה זהירה יותר של האלגוריתם, זמן הריצה הוא $\mathcal{O}(m \cdot \sqrt{m} \cdot \log n \cdot \log U)$.

עדיין נשארו כמה בעיות פתוחות בנושא:

- האם אפשר להריחב את הרעיון הזה לבעיית הזרימה המקסימלית הכללית?
- האם אפשר להוסיף גם עלויות לקשתות, ולקבל אלגוריתם לזרימה בעלות מינימלית בזמן של $\mathcal{O}(m \cdot \sqrt{m} \cdot \log n \cdot \log(n \cdot C) \cdot \log(n \cdot U))$?

פרק 7

חיבוריות בגרף דינאמי

7.1 הקדמה

נניח שקיים גרף שאינו קבוע, המאפשר הכנסה והוצאה של קשתות (לא מכוונות ולא ממושקלות). נרצה לדעת אם שני קודקודים v ו־w כלשהם נמצאים באותו רכיב קשירות של הגרף.

הבעיה מעניינת בעיקר בגלל ההוצאה של הקשתות. בלעדיה, הבעיה דומה ל־Union find, וקל מאוד לרדת מ־$\mathcal{O}(\log n)$ זמן.

הפעולות שמעניינות אותנו הן:

- `Insert` (v, w): מכניסה קשת בין הצמתים v ו־w.
- `Delete` (v, w): מוחקת את הקשת בין הצמתים v ו־w.
- `Connected` (v, w): בודקת אם v ו־w מחוברים בגרף, כלומר אם קיים מסלול בגרף מ־v ל־w.

7.2 רעיון למימוש

נתחזק יער של עצים פורשים. כל קשת בגרף יכולה להיות אדומה (כלומר, חלק מהעץ) או שחורה (לא בעץ).

כשמוסיפים קשת לגרף, מוסיפים אותה גם לעץ אם היא מחברת שני רכיבי קשירות. כשבודקים אם שני צמתים נמצאים באותו רכיב קשירות, בודקים אם הם באותו העץ. כשמוחקים קשת שחורה, לא צריך לעשות כלום. הבעיה היא במקרה שבו מוחקים קשת כחולה.

יכול להיות שהקשת הכחולה שאותה מחקנו היא זו שמחברת בין שני רכיבי קשירות, ואז זה בסדר שהעצים יתפצלו לשני עצים שונים. אבל יכול להיות שיש קשת שחורה שמחברת בין שני העצים האלה, ונרצה לדעת עליה בצורה יעילה כדי להוסיף אותה ליער הפורש במקום הקשת שמחקנו זה עתה.

בינתיים, אנחנו צריכים Abstract Datatype שיספק את הפעולות הבאות:

- `Link` (v, w)

הפעולה חוקית רק אם v ו־w נמצאים בעצים שונים ביער. הפעולה מוסיפה קשת בין v ל־w בעץ.

- `Cut` (v, w)

הפעולה חוקית רק אם v ו־w מחוברים בקשת ביניהם. הפעולה מוחקת את הקשת ביניהם.

- `Findtree` (v)

הפעולה מחזירה את העץ שבו הצומת v נמצא.

יכול להיות שנצטרך עוד פעולות. נראה בהמשך.

פתרון טריוויאלי לבעיה של מחיקת קשת כחולה הוא סריקת כל הקשתות השחורות, ובדיקה אם אחת מהן מחברת בחזרה את שני העצים שנותקו. זה פתרון נאיבי, והוא מאוד יקר בזמן ריצה. אבל יכול להיות שאנחנו יכולים לקבל בסריקה כזאת אינפורמציה גם לפעמים הבאות, ולהשתמש במידע הזה בפעם הבאה כדי לחסוך זמן.

את המידע הזה נקודד בקשתות. לכל קשת תהיה רמה – מספר אי־שלילי שלם. בתחילת האלגוריתם (או כשמוסיפים את הקשת), הרמה שלה תהיה 0.

טענה 95: יהי F_i תת היער המורכב מרשתות ברמה לפחות i. אזי $F_{i+1} \subseteq F_i$.

נניח שזורקים קשת (אדומה) מ־F_0 ברמה ℓ. אזי היא שייכת לאיזה עץ T ב־F_ℓ. אם יש קשת חליפית (x, y) מרמה גדולה מ־ℓ, אזי $x, y \in T$ (כצמתים), כי אחרת מקבלים סתירה למקסימליות של העץ הפורש. האלגוריתם יחפש חיבור חליפי בסדר רמות יורד. מההוצאה של הקשת (v, w), האלגוריתם מקבל שני עצים – T_v ו־T_w. נניח בה"כ כי $|T_v| \leq |T_w|$.

נגדיל את הרמה של כל הקשתות ב־T_v ב־1. אחר כך מגדילים ב־1 את הרמה של כל הקשתות השחורות שעוברים עליהן (הסמוכות ל־T_v) שאינן מחברות בין T_v ל־T_w. אם אין קשת מחליפה ברמה ℓ, מגדילים את כל הקשתות האדומות ברמה $\ell - 1$ ב־T_v לרמה ℓ, ואז מחפשים קשתות שחורות ברמה $\ell - 1$ שהן חליפיות ל־(v, w). אם לא מוצאים ברמה $\ell - 1$, ממשיכים לרמה $\ell - 2$. אם מגיעים לרמה 0 ועדיין לא מוצאים קשת חליפית, אז (v, w) הייתה הקשת היחידה שחיברה בין שני רכיבי קשירות, והסרתה ניתקה אותם.

הבהרה: בצעד ה־i סורקים קשתות שחורות ברמה ה־i בדיוק, שחלות על הצד הקטן של T_i.

כמה אינווריאנטות שתמיד מתקיימות ושומרות על נכונות האלגוריתם:

1. היער הוא יער פורש מינימלי ביחס לרמה של הקשתות.

   לכן, לכל קשת שחורה, כל מסלול בעץ בין שני הקודקודים שלה מורכב מקשתות שרמתן גדולה או שווה לרמת הקשת השחורה. אחרת, אפשר היה להחליף קשת אדומה בקשת השחורה ולקבל עץ פורש "גדול" יותר. לכן, אם אנחנו מוחקים קשת אדומה ברמה ℓ, הקשת המחליפה היא ברמה לכל היותר ℓ.

2. הגודל של כל עץ ב־F_i הוא לכל היותר $\frac{n}{2^i}$, ולכן יש לכל היותר $\log n$ רמות.

   למה האינווריאנטה הזאת נשמרת? אם מחקנו את הקשת (v, w) מרמה ℓ, אזי מהאינווריאנטה (בהנחה שהיא התקיימה עד כה), הגודל של העץ T שהכיל את (v, w) לא היה גדול מ־$\frac{n}{2^\ell}$. בה"כ $|T_v| \leq |T_w|$, ולכן $|T_v| \leq \frac{n}{2^{\ell+1}}$. לכן, לכל היותר $\frac{n}{2^{\ell+1}}$ קשתות עולות לרמה $\ell + 1$.

אלגוריתם 1.7 מימוש הפעולות לבעיית החיבוריות בגרף דינאמי

```
1: procedure INSERT(v, w)
2:     if FINDTREE(v, F_0) ≠ FINDTREE(w, F_0) then LINK(v, w, F_0)
3:     else add a non-tree edge of level 0 to v and w
4: end procedure

5: function CONNECTED(v, w)
6:     return FINDTREE(v, F_0) = FINDTREE(w, F_0)
7: end function

8: procedure DELETE(v, w)
9:     ℓ ← LEVEL(v, w)
10:    if (v, w) is a non-tree edge then delete it from v and w in F_ℓ
11:    else
12:        for all i = ℓ, ℓ − 1, ..., 0 do CUT(v, w, F_i)
13:        Find a replacement edge as described above
14:        if a replacement edge (x, y) was found in level k ≤ ℓ then
15:            for all i ∈ {0, 1, ..., k} do LINK(x, y, F_i)
16:        end if
17:    end if
18: end procedure
```

7.3 מימוש יותר מדוייק

נתחזק כל F_i כאובייקט נפרד (למשל בתוך עצים דינאמיים). הקשתות השחורות ברמה ℓ נמצאות בתוך העותקים של הצמתים ב־F_ℓ. כלומר, אם הקשת (v,w) היא שחורה ברמה ℓ, אזי העותקים של v ו־w ב־F_ℓ יצביעו לקשת (v,w).

המימוש המדוייק מופיע באלגוריתם 1.7.

נוספו לנו עוד 3 פעולות ל־Abstract Datatype שבו נשתמש:

1. מציאת קשת אדומה ברמה ℓ שנמצאת בצומת $v \in F_\ell$.
2. מציאת קשת שחורה ברמה ℓ שסמוכה לעץ $T \in F_\ell$.
3. הוספה ומחיקה של קשת שחורה ברמה ℓ שסמוכה לצומת $v \in T \in F_\ell$.

7.4 ניתוח

נניח שכל פעולה על ה־Abstract Datatype שלנו מתבצעת בזמן $\mathcal{O}(\log n)$.

משפט 96: סדרה של m פעולות על המבנה של חיבוריות דינאמית לוקחת $\mathcal{O}\left(m \cdot \log^2 n\right)$ זמן. כל פעולה לוקחת $\mathcal{O}\left(\log^2 n\right)$ זמן.

הוכחה: `Insert` (v,w) לוקחת $\mathcal{O}(\log n)$ זמן, ועוד הזמן הדרוש להגדלת רמות של קשתות. כל הגדלה לוקחת $\mathcal{O}(\log n)$ זמן, ויש $\log n$ רמות, ולכן הזמן הכולל לפעולה הוא $\mathcal{O}\left(\log^2 n\right)$.

באופן דומה, גם Delete(v,w) לוקח $\mathcal{O}\left(\log^2 n\right)$ זמן.
באשר ל־Connected(v,w): יש רק שאילתא פשוטה מה־Abstact Datatype ב־$\mathcal{O}\left(\log n\right)$ זמן. ■

7.5 מימוש ה־Abstract Datatype בצורה יעילה

אפשר לעשות שינויים קלים לעצים דינאמיים כדי לתמוך בכל הפעולות שאנחנו רוצים, אבל יש דרך פשוטה יותר: Euler Tour Tree.

הרעיון ב־Euler Tour Tree הוא שהעץ מיוצג על ידי רשימה של קשתות, והקשתות מסודרות לפי סדר ש"מטייל" על הצמתים. כל קשת מופיעה פעמיים, פעם אחת בכל כיוון.

קל לתמוך בפעולות Link ו־Cut בצורה יעילה עם רשימה מקושרת של קשתות. אבל מה עם מציאת העץ שבו נמצא צומת? כדי לעשות את זה בצורה יעילה, נייצג את הרשימה בעץ חיפוש.

7.6 עץ פורש מינימלי בגרף משתנה

נתון גרף עם n צמתים ו־m קשתות. לכל קשת יש עלות. צריך לתחזק עץ פורש מינימלי תוך תמיכה בהורדת קשתות מהגרף עם Delete.

נראה איך לעשות את זה ב־$\mathcal{O}\left(m \cdot \log^2 n\right)$ זמן. יהיה קל להוסיף לאלגוריתם שנראה עכשיו פעולת Insert בלי לפגוע בסיבוכיות.

האלגוריתם מאתחל את היער הפורש להיות היער הפורש המינימלי. כל הקשתות מתחילות ברמה 0. כשמוציאים קשת, נחפש את הקשת החליפית במנגנון שראינו קודם (עם הרמות). כשסורקים את הקשתות המועמדות ברמה מסויימת, עושים זאת לפי עלות עולה.

כדי להראות את הסיבוכיות, חשוב להראות את האינווריאנטה הבאה: בכל מעגל C בגרף יש קשת שאינה ביער הפורש עם עלות מקסימלית ורמה נמוכה ביותר. פעולת מחיקת הקשתות שומרת על האינווריאנטה הזאת: מה קורה אם אנחנו מעלים רמה של קשת או מוסיפים אותה לעץ הפורש?

נרצה להראות שברגע שהאלגוריתם בוחן את הקשת (v,w), האינווריאנטה נשמרת. נתבונן במעגל C שבו (v,w) היא הקשת היחידה המקסימלית בעלות ומינימלית ברמה i, ולא בעץ. כל הקשתות על C שחלות על T_v (פרט ל־(v,w)) וקטנות בעלות מ־(v,w) הן כבר ברמה גדולה ממש מ־i, או שהן מלכתחילה ברמה גדולה מ־i או ברמה i וכבר סרקנו אותן, כי העלות שלהן קטנה מהעלות של (v,w), ולא חיברו את T_v ו־T_w, ולכן עלו ברמה.

לכן, המעגל C אינו יוצא מהרכיב שנפרש על ידי T_v. אם יוצאים ממנו, אז יש לפחות שתי קשתות מ־C שחלות על T_v. כלומר, יש עוד קשת $(x,y) \neq (v,w)$ על המעגל שעוזבת את T_v. (x,y) היא מרמה גדולה ממש מ־i, ומחברת את T_v עם T_w, ולכן זוהי סתירה, כי היא הייתה צריכה להתגלות קודם.

למה 97: הקשת החליפית בעלות הנמוכה ביותר היא בעלת הרמה הגבוהה ביותר.

פרק 8

בעיית שימור הסדר

8.1 הגדרת הבעייה

נרצה לתחזק מבנה נתונים שיתנהג כמו רשימה L עם הפעולות הבאות:

- $\texttt{Insert}(x,y)$: מכניס את y מייד אחרי x ברשימה.
- $\texttt{Delete}(x)$: מוחק את x מהרשימה.
- $\texttt{Order}(x,y)$: בודק אם x נמצא לפני y ברשימה.

8.2 ניסיון ראשון

פסאודו־קוד של הניסיון מתואר באלגוריתם 1.8.

נתחיל בכך שלכל איבר ברשימה ניתן תווית (Label), לפי הסדר (כלומר מספר טבעי שיתאים לו). בכל פעם שמכניסים איבר לרשימה, התווית שתתאים לו תהיה ממוצע התוויות של האיברים הסמוכים אליו. המחיקה לא דורשת הרבה, אבל קל מאוד לדעת איזה איבר מגיע ראשון, לפי הגודל של התוויות.

כך למעשה מימשנו את כל הפעולות בזמן קבוע, לכאורה. הבעיה במימוש הזה היא שהייצוג של התוויות יכול להיות בגודל אינסופי (למשל אם תמיד מכניסים איברים אחרי האיבר הראשון, בכל פעם צריך סיבית נוספת כדי לייצג את התווית), וכדי לטפל בתוויות בגודל משתנה, סיבוכיות הזמן כבר לא קבועה.

8.3 ניסיון שני

8.3.1 הרעיון הכללי

נצמיד תווית שלמה בין 1 ל־M לכל איבר ברשימה. האיברים ישבו ברשימה משורשרת, כמו קודם. הרשימה למעשה תייצר לנו עץ בינארי מלא, שהוא יהיה עץ לוגי (לפעמים יהיה קל יותר לחשוב על הרשימה במונחים של עץ), אבל בזיכרון נייצג רק את הרשימה. למעשה, במקום להדביק תווית לכל איבר ברשימה, נחזיק מערך בגודל M, והאיברים ברשימה יהיו איברים במערך. לא כל איבר במערך חייב להכיל איבר מהרשימה.

אלגוריתם 1.8 ניסיון ראשון למימוש שימור סדר

```
1: procedure INSERT(x, y)
2:     if x is the last element in the list then INDEX(y) ← INDEX(x) + 1
3:     else
4:         z ← PREDECESSOR(x)
5:         INDEX(y) ← (INDEX(x)+INDEX(z))/2
6:     end if
7: end procedure

8: procedure DELETE(x)
9:     Delete x from the list
10: end procedure

11: function ORDER(x, y)
12:     return INDEX(x) < INDEX(y)
13: end function
```

איך נבחר את M? נקבע $T \in (1,2)$ שרירותי (נראה בהמשך איך הוא משפיע על הסיבוכיות). נשייך צפיפות לפי T: 1 לעלים, $\frac{1}{T}$ לרמה שמעליהם, $\frac{1}{T^2}$ לרמה שמעליהם, וכן הלאה. נבחר M גדול מספיק כך שהצפיפות "האידיאלית" תהיה טובה.

8.3.2 הגדרות

הגדרה 98 (צפיפות): הצפיפות (בפועל) של צומת v בעץ היא יחס בין מספר העלים המכילים איבר בתת־העץ של v לבין כמות העלים הכוללת בתת־העץ של v.

הגדרה 99 (צפיפות אידיאלית): לצומת ברמה i יש צפיפות אידיאלית של $\frac{1}{T^i}$.

נבחר את M כך שאם יהיו ברשימה $2 \cdot n$ איברים, אזי הצפיפות בפעול בשורש לא תחרוג מהצפיפות האידיאלית של השורש. כלומר:

$$\frac{1}{T^{\log M}} = \frac{2 \cdot n}{M} \tag{8.1}$$

לכן, נרצה את ה־M המינימלי שיקיים $\frac{1}{T^{\log M}} \geq \frac{2 \cdot n}{M}$, ואם אפשר שהשוויון יתקיים.
נתחיל פאזה חדשה כאשר האינווריאנטה $M/2 \leq n \leq 2 \cdot M$ תפסיק להתקיים, כאשר n הוא מספר האיברים ברשימה.
בתחילת פאזה, נבחר M שמקיים את (??), ונמספר את כל האיברים במרווחים שווים במרחב $\{1, \ldots, M\}$.

נפתור את (??):

$$\begin{aligned} T^{-\log M} = \frac{1}{T^{\log M}} &= \frac{2\cdot n}{M} \\ -\log M \cdot \log T &= \log(2\cdot n) - \log M \\ \log M \cdot (1-\log T) &= \log(2\cdot n) \\ \log M &= \frac{\log(2\cdot n)}{1-\log T} \\ M &= 2^{\frac{\log(2\cdot n)}{1-\log T}} \end{aligned}$$

מכיוון ש־$T \in (1,2)$ קבוע, $1-\log T$ חסום, ולכן:

$$M = 2^{\frac{\log(2\cdot n)}{1-\log T}} \sim 2^{\log(2\cdot n)} = 2\cdot n \sim n$$

למה 100: כל תווית ניתנת לייצוג בעזרת $\mathcal{O}(\log n)$ ביטים.

הוכחה: כל תווית מיוצגת בעזרת $\mathcal{O}(\log M)$ ביטים. כפי שראינו, $M \sim n$, ולכן כל תווית ניתנת לייצוג בעזרת $\mathcal{O}(\log n)$ ביטים. ■

8.3.3 תיאור האלגוריתם

הבעיה האמיתית היא בביצוע של `Insert`. הביצוע של שאר הפעולות ברור. ניתן לראות פסאודו־קוד למימוש באלגוריתם 2.8.

כאשר מבצעים $\texttt{Insert}(x,y)$, אם יש מקום פנוי אחרי x, אז שמים שם את y. אחרת, עולים בעץ עד שמגיעים לרמה שהצפיפות של תת־העץ שלה נמוכה מהצפיפות האידיאלית, ואז מסדרים מחדש את כל האיברים בעץ העץ הזה במרווחים שווים. עכשיו יהיה מקום להכניס את y אחרי x, כרגיל.

הערה 101: העץ לא באמת קיים. אנחנו מבצעים את חישובי הצפיפות על המערך, והעץ קיים רק ברמה הרעיונית־לוגית.

הערה 102: אם עברנו את הצפיפות האידיאלית בשורש העץ, אז אנחנו "מייצרים" עץ חדש, שגדול כפליים מהעץ המקורי, ומרווחים מחדש את כל הצמתים במרווחים שווים. כנ"ל אם יש לנו פחות מ־$M/2$ איברים.

8.3.4 ניתוח סיבוכיות

נניח שעשינו סידור מחדש של האיברים בתת־העץ של x. כדי שנבצע שוב סידור מחדש בתת־העץ של x, הצפיפות באחד מילדיו צריכה לעלות מעל $\frac{1}{T^{i-1}}$ (בסוף הסידור הראשון, הצפיפות הייתה מתחת ל־$\frac{1}{T^i}$). לכן, בין שני הסידורים האלה, מספר האיברים שהכנסו לרשימה הוא:

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

אלגוריתם 2.8 פתרון מורכב יותר לבעיית שימור הסדר

```
1: procedure INSERT(x, y)
2:     Check if there is a need to start a new phase (n + 1 ≥ 2 · M)
3:     z ← SUCCESSOR(x)
4:     if x and z are not consequtive then LABEL(y) ← any label between LABEL(x) and LABEL(z)
5:     else
6:         Find the lowest ancestor of x such that the density of its subtree is below T
7:         Relabel the elements in this subtree
8:         LABEL(y) ← any label between LABEL(x) and LABEL(z)
9:     end if
10: end procedure

11: procedure DELETE(x)
12:     Mark A[LABEL(x)] as free
13:     Check if there is a need to start a new phase (n ≤ M/2)
14: end procedure

15: function ORDER(x, y)
16:     return LABEL(x) < LABEL(y)
17: end function
```

העבודה שמושקעת בסידור מחדש של תת־עץ בגובה i היא $\mathcal{O}\left(\frac{2^i}{T^i}\right)$. לכן, כמות העבודה שאפשר לחייב על כל הכנסה במקום על הסידור מחדש היא:

$$\frac{\frac{2^i}{\cancel{T^i}}}{2^{i-1} \cdot \frac{T-1}{\cancel{T^i}}} = \frac{2}{T-1}$$

כמה רמות מחייבות את ההכנסה של הצומת x? לכל היותר $\log M = \mathcal{O}(\log n)$.

פרק 9

זיהוי מעגלים מצטברים

מתחילים עם גרף עם n קודקודים, בלי קשתות. מכניסים m קשתות מכוונות, אחת אחת (לפי סדר כלשהו). אם נסגר מעגל, צריך לדווח על כך ולהפסיק.

פתרון נאיבי יכול להיות שימוש ב־BFS או DFS מהצומת w כשמכניסים את הקשת (v,w). אם החיפוש הגיע ל־v, אז נסגר מעגל. זמן העבודה הכולל של האלגוריתם יהיה $\mathcal{O}\left(m^2\right)$.

9.1 מיון טופולוגי של הגרף

הצעה יותר טובה היא שימוש במיון טופולוגי של הגרף, ושמירה עליו ככל שהאלגוריתם מתקדם. הסיבוכיות של הפתרון היא $\mathcal{O}\left(m \cdot n\right)$.

בעזרת המיון הטופולוגי, אפשר לדעת אם נסגר מעגל. אם מכניסים את הקשת (v,w) לגרף, ו־w נמצא לפני v במיון הטופולוגי, אין בכך כדי לומר דבר על סגירת מעגל (התנאי לא הכרחי ולא מספיק). צריך להתקדם קדימה מ־w (לפי הקשתות והמיון הטופולוגי), ולראות אם מגיעים ל־v, או שעוברים אותו ומדלגים עליו. אם הגענו ל־v, אז נסגר כאן מעגל. אחרת, צריך לשנות את המיון הטופולוגי כך ש־v יבוא לפני w.

זה אפשרי, כי מיון טופולוגי הוא לא יחיד. יכול היה גם לקרות מצב שבו v היה מופיע לפני w במיון הטופולוגי. מכיוון שלא הגענו ל־v כשהתחלנו חיפוש דרך w, אפשר להזיז את w וכל הצמתים הנגישים מ־w להיות מייד אחרי v במיון הטופולוגי, והמיו הטופולוגי יישאר חוקי.

9.2 אנליזה

הגדרה 103: נאמר שקשת (x,y) וצומת v הם קשוריס ($Related$) אם קיים מסלול בגרף שמכיל גם את v וגם את (x,y).

אם לא מצאנו מעגל, בחיפוש שלנו, אז v לא היה קשור לכל קשת (x,y) שעברנו דרכה במהלך החיפוש שהתחיל ב־w. אחרי שהוספנו את הקשת (v,w), v הפוך לקשור לכל אחת מהקשתות האלה.

לכן, בכל הוספת קשת לגרף, הוספנו לפחות זוג אחד של קשת וצומת קשורים. ליתר דיוק, בכל פעם שאנחנו סורקים את הקשת (x,y), אם לא נסגר מעגל, אנחנו קושרים את

(x,y) עם v. לכן, ניתן לחייב כל צעד בחיפוש על ידי קשירה של קשת וצומת. מכיוון שיש $m \cdot n$ זוגות של קשת וצומת, זמן הריצה של האלגוריתם הוא $\mathcal{O}(m \cdot n)$.

9.3 חיפוש דו־כיווני

חיפוש דו־כיווני (Two-way search) הוא שינוי קל של האלגוריתם הקודם, רק שהפעם במקום לחפש רק על הקשתות היוצאות מ־w, אנחנו נעבור על זוגות תואמים (*Compatible Pairs*) של קשתות מהצורה (a,b) ו־(x,y), כאשר (a,b) היא קשת שהתקבלה מהליכה אחורה על קשתות הנכנסות ל־v, ו־(x,y) היא קשת שהתקבלה מהליכה קדימה על קשתות היוצאות מ־w, כך ש־$a < y$ לפי המיון הטופולוגי.

נגדיר שתי קבוצות:

F קבוצת הצמתים שהחיפוש קדימה מ־w מבקר בהם.

B קבוצת הצמתים שהחיפוש אחורה מ־v מבקר בהם.

אזי $B \cap F = \emptyset$, כי אחרת היה נסגר מעגל (ואז זה לא מעניין). נסדר את F לפי המיון הטופולוגי, ואת B בסדר הפוך מהמיון הטופולוגי.

בכל שלב, יהיו לנו צומת נוכחי u ב־F (המאותחל להיות w), וצומת נוכחי y ב־B (המאותחל להיות v). נסרוק זוג קשתות (שלא נצבעו עדיין) – (v,z) ו־(x,y). אם $z \in B$ או $x \in F$ או $z = x$, אז נסגר מעגל. אחרת, נצבע את z לירוק ואת x לכתום. קידום הצומת הנוכחי ב־F נעשה כאשר סרקנו את כל הקשתות היוצאות ממנו. באופן דומה, קידום הצומת הנוכחי ב־B נעשה כאשר סרקנו את כל הקשתות הנכנסות אליו. נעצור כאשר הצומת הנוכחי מ־B נמצא לפני הצומת הנוכחי מ־F במיון הטופולוגי, ואז נאמר שלא גילינו מעגל.

תרגיל 104: אם לא גילינו מעגל, אז לא נסגר מעגל בעקבות הוספת (v,w).

אם לא מצאנו מעגל, כל זוג קשתות שנסרקו הן תואמות. הסידור מחדש מתבצע באופן הבא: נסמן ב־f את הצומת הנוכחי של F ברגע שעצרנו. את הקודקודים בין w ל־f נזיז להיות סמוכים מייד לפני w, תוך שמירה על סדר טופולוגי ביניהם. צריך להוכיח שהסידור נותן סידור טופולוגי של הגרף עם (v,w).

משפט 105: כאשר מוסיפים סדרה של m קשתות, החיפוש הדו־כיווני עובר על $\mathcal{O}\left(m^{3/2}\right)$ קשתות.

הוכחה: נאמר שהקשתות (a,b) ו־(x,y) קשורות אם יש מסלול מכוון שמכיל את שתיהן. נניח שלא נוצר מעגל. אזי כל זוגות הקשתות שעברנו עליהם במהלך החיפוש לא היו קשורות לפני הוספת הקשת, אבל הן קשורות אחרי הוספת הקשת.

נאמר שההכנסה של הקשת ה־i גררה חיפוש בין k_i זוגות קשתות. במילים אחרות, בין k_i קשתות הפכו לקשורות בעקבות ההכנסה של הקשת ה־i. נאמר שההכנסה ה־i היא הכנסה קטנה אם $k_i \leq \sqrt{m}$, ונאמר שהיא הכנסה גדולה אם $k_i > \sqrt{m}$.

מספר ההכנסות הקטנות חסום על ידי m (זה המקרה הגרוע, שבו כל ההכנסות הן קטנות). במקרה זה, מספר הקשתות שנסרקו בכל ההכנסות הקטנות חסום על ידי $m \cdot \sqrt{m} = m^{3/2}$.

נאמר כי $j_1, j_2, \ldots$ הן ההכנסות הגדולות. אזי ההכנסה ה־j_i יצרה עוד $k_{j_i}^2$ קשתות קשורות. מצד אחד, מספר הקשתות שיכולות להיות קשורות חסום על ידי מספר זוגות

הקשתות, כלומר:

$$\sum_i k_{j_i}^2 \leq \binom{m}{2}$$

מצד שני, $k_{j_i} > \sqrt{m}$, ולכן:

$$\sqrt{m} \cdot \sum_i k_{j_i} = \sum_i \sqrt{m} \cdot k_{j_i} < \sum_i k_{j_i} \cdot k_{j_i} = \sum_i k_{j_i}^2$$

משילוב של שתי המשוואות, נקבל:

$$\sum_i k_{j_i} \leq m \cdot \sqrt{m} = m^{3/2}$$

■

מסקנה 106: מכיוון שהזמן שלוקח לעבור על זוג קשתות הוא $\mathcal{O}(\log n)$ (כי F ו־B מיוצגים כערימות), זמן הריצה הכולל של האלגוריתם הוא $\mathcal{O}\left(m^{3/2} \cdot \log n\right)$.

הערה 107: אפשר לקבל זמן ריצה טוב יותר של $\mathcal{O}\left(m^{3/2}\right)$, אבל לא נראה את זה כאן.

9.4 חיפוש חד־כיווני לגרפים צפופים

לכל צומת u, נאמר ש־$\text{size}(u)$ הוא מספר הצמתים מהם יש מסלול ל־u (כולל). נחזיק $k(u)$ כך ש:

$$k(u) \leq \text{size}(u) \tag{9.1}$$

כמו כן, נדרוש שלכל קשת (u, v):

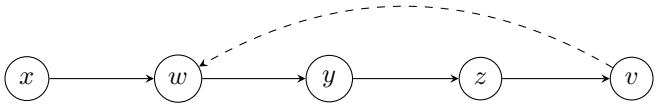
$$k(u) < k(v) \tag{9.2}$$

אזי $k(w)$ הוא מספור טופולוגי חלש.

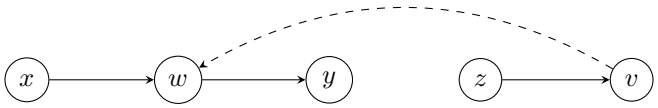
כאשר מכניסים את הקשת (v, w), אז כמו קודם. אם $k(v) < k(w)$, בוודאות לא נסגר מעגל (מתכונה (??)). אחרת, ייתכנו שני מקרים. באחד מהם נסגר מעגל (כפי שניתן לראות באיור 9.1), ובשני לא (כפי שניתן לראות באיור 9.2). נרצה לדעת להבחין ביניהם.

במקרה הזה, נתחיל סריקה מ־w על מנת לוודא שאין מעגלים (בעצם אנחנו רוצים לבדוק אם v נגיש מ־w). נעצור את החיפוש כל פעם שנגיע לצומת u כך ש־$k(u) \leq k(v)$. אם לא נמצא מעגל אחרי הוספת (v, w), נגדיל את ערכי ה־k של קבוצת קודקודים כך שהמיון הטופולוגי יתקיים גם עם הקשת (v, w).

באופן אינטואיטיבי, בגלל שלכל $u \in V$, $k(u) \leq \text{size}(u) \leq n$, ניעזר בזה כדי לחסום את העבודה. כדי לעדכן את ערכי ה־k, כל צומת u מחזיק את הקשתות שיוצאות ממנו בערימה המסודרת לפי ערכי ה־k של הצומת אליו הקשת מגיעה. נוציא מהערימה בזה אחרי זה את הצמתים עם k שקטן או שווה לערך החדש של x, נעדכן את ה־k שלי ונמשיך רקורסיבית את החיפוש מכל אחד מהצמתים שעודכנו. ברגע ש־$k(x)$ גדל בגלל סריקת קשת מ־y ל־x, לא מעדכנים את המפתח של x בערימות של צמתים שמצביעים ל־x, פרט ל־y.



איור 9.1: הוספת הקשת (v,w) סוגרת מעגל כאשר $k(v) \geq k(w)$



איור 9.2: הוספת הקשת (v,w) אינה סוגרת מעגל כאשר $k(v) \geq k(w)$

לכן, הדבר הנכון לומר הוא שבערימה של x, לא נמצא ה־k המשוייק של השכן y, אלא ה־k בפעם האחרונה ש־x "הסתכל" על y. הערכים הערימות אלה (אולי) קטנים מהערכים האמיתיים.

נסביר שוב על החיפוש: כשהחיפוש מגיע לצומת x (שערכו התעדכן), $k(x) > k^P(x)$. מוציאים מהערימה של x את כל השכנים y (שמחוברים ל־x בקשת היוצאת מ־x), כך שהמפתח של y בערימה של x הוא לכל היותר $k(x)$. מסתכלים מה הערך האמיתי של $k(y)$. אם $k(y) > k(x)$, מעדכנים את המפתח של y בערימה של x להיות $k(y)$. אחרת, מעדכנים את $k(y)$ האמיתי (נראה בהמשך מה יהיה הערך החדש של $k(y)$), ומעדכנים אותו גם בערימה של x. בשלב הזה, החיפוש יעבור רקורסיבית ל־y.

למה 108: אם יש קשת מ־x_j ל־v, ולכל i, $size(x_i) \geq s$, אזי $size(v) \geq s+j$.

הוכחה: במקרה שבו $\text{size}(v)$ הכי קטן, יש קשת מכל אותם $s-1$ צמתים לכל x_i. במקרה זה, כל ה־$s-1$ צמתים נספרים פעם אחת, עוד j צמתים מסוג x_i, וגם v נספר. לכן:

$$\text{size}(v) \geq s-\not{1}+j+\not{1} = s+j$$

■

איך נשתמש בזה? כל צומת v יזכור רשימה של צמתים y_i כך שקיימת קשת (y_i,v) במבנה נתונים המסודר לפי $k(y_i)$ (כאשר $k(y_i)$ מעודכן לפי הפעם האחרונה ש־y_i גרם לעדכן של $k(v)$).

מלמה 108, לכל j, $\text{size}(v) \geq k(y_j)+j$. לכן נקבע:

$$k(v) = \max_j \{k(y_j)+j\}$$

לכן, בכל פעם שנרצה לעדכן את הערך של $k(v)$, נקבע אותו להיות $\max_j \{k(v), k(y_j)+j\}$. עדיין נותרו לנו כמה נקודות פתוחות:

- באיזה מבנה נתונים עלינו להשתמש כדי לשמור את ה־k־ים של הקשתות הנכנסות בכל צומת?

  אנחנו רוצים להחזיק קבוצת מספרים $k(y_1) \geq k(y_2) \geq \cdots$ תחת הכנסות, הוצאות ושאילתות שמוצאות את $\max_j \{k(y_j) + j\}$. בכל צומת v של העץ, נזכור את מספר ה־$k(y_i)$־ים שיש בתת־העץ שלו, וכן את $\max_{y \in T_v} \{k(y) + \#\text{precede } y \text{in } T_v\}$. בפרט, בשורש יושב ה־max הנדרש מהשאילתה.

  קל לראות שאם z הוא האבא של x ו־y, והאינפורמציה ב־x וב־y מעודכנת, ניתן להעדכן גם את z בזמן קבוע. זה מאפשר לנו לבצע סיבובים בזמן קבוע, ולכן להכניס ולהוציא איברים בזמן לוגריתמי עם עצים מאוזנים. ה־max ב־z הוא הגדול מבין ה־max של y וה־max של x שמחובר עם הגודל של y.

- כמה עדכונים כאלה יש במקרה הגרוע?

  יש לכל היותר n עדכונים שבאמת מעדכנים את $k(v)$, ולכן בסך הכל $\mathcal{O}(n^2)$.

  נבדוק כמה עדכוני שווא כאלה יש לנו (שבהם y רואה $k(v)$ שאינו מעודכן, ובפועל $k(v)$ לא מתעדכן). יהי $k'(v) \leq k(y)$ הערך ש־y חושב שהוא של v, ויהי $k(v) > k(y)$ הערך האמיתי. נמפה את העדכון הזה לאינטרוול $[k'(v), k(v)[$, והוא ימופה לצומת בעץ בינארי מאוזן, שהעלים בו הם כל הערכים האפשריים ל־$k(v)$ (שזה מ־1 עד n). הצומת הזה יהיה ה־Lowest common ancestor של $k'(v)$ ו־$k(v)$.

  כמה אינטרוולים יכולים להתמפות לצומת בעץ? נסתכל על צומת עם 2^i עלים תחתיו. יש לכל היותר עדכון אחד מצומת מסויים y, ויש לכל היותר 2^i צמתים בתת־העץ של הצומת. לכן, אחרי לכל היותר 2^i נסיונות לעדכון של v עם $k(y) > k'(v)$, $k(v)$ כבר לא יכול להיות באותו תת־עץ.

  לכן, יש בסך הכל $\mathcal{O}(n \cdot \log n)$ עדכוני שווא, ו־$\mathcal{O}(n^2 \cdot \log n)$ עדכונים באופן כללי בריצת האלגוריתם.

לכן, בסך הכל, זמן הריצה של האלגוריתם הוא $\mathcal{O}(n^2 \cdot \log^2 n)$.

פרק 10

Suffix Trees

10.1 Trie

Trie מוגדר מאוסף של מחרוזות, וכל מסלול בו מהשורש מתאים לאחת המחרוזות, וכל קשת מייצגת אות. אין שתי קשתות שיוצאות מותו הצומת ומייצגות את אתה האות. דוגמה ל־Trie ניתן למצוא באיור 10.1.

ניתן גם לדחוס Trie, במקרים שבהם יש מסלולים שלמים ללא פיצולים. ניתן למצוא דוגמה לכך באיור 10.2.

10.2 Suffix Tree

הגדרה 109 (Suffix Tree): תהי מחרוזת s. ה־*Suffix Tree* של s הוא Trie דחוס של כל הסיומות (Suffixes) של s.

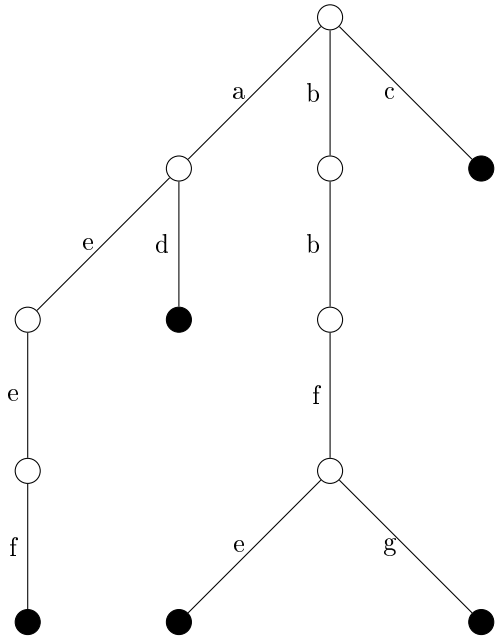
כדי שהסיומות האלה יהיו מדוייקות, נוסיף לסוף המחרוזת תו חדש, למשל \$, בסוף של s.

האלגוריתם הטריוויאלי לבניית Suffix Tree מכניס את כל הסיומות של s לפי הסדר (מהגדולה לקטנה) לתוך Trie. לוקח $\mathcal{O}\left(n^2\right)$ זמן לבנות ככה Suffix Tree.

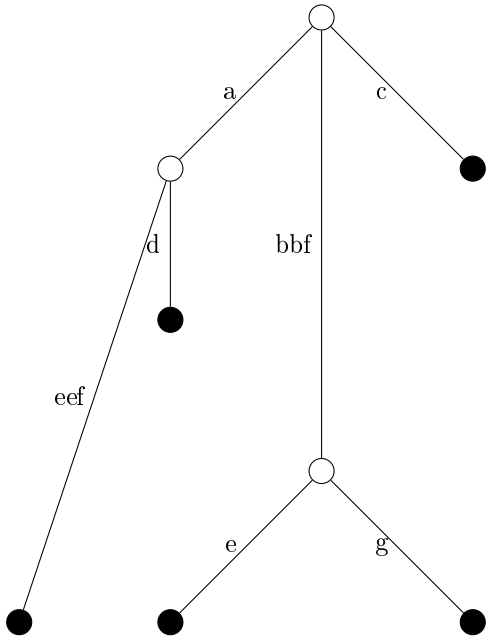
אפשר לעשות את זה ב־$\mathcal{O}(n)$ זמן (כפי שנראה בהמשך). אבל איך זה ייתכן? הרי הגודל של Suffix Tree הוא $\mathcal{O}\left(n^2\right)$. במקום לכתוב על הקשתות של ה־Trie את כל תת־המחרוזת שמתאימה לקשת, נקודד אותה בעזרת שני מספרים, שהם למעשה האינדקסים של תת־המחרוזת בתוך המחרוזת הגדולה. באופן הזה, נצטרך $\mathcal{O}(n)$ זיכרון כדי לקודד את ה־Trie, ואז (אולי) אפשר לרדת לזמן לינארי.

איזה שימושים יש ל־Suffix Tree? למשל חיפוש מחרוזת קצרה בתוך מחרוזת גדולה. בהינתן טקסט T, $|T| = n$, נצטרך לעשות לו קצת עיבוד מקדים, כדי שכשנקבל מחרוזת P, $|P| = m$, נוכל להחליט במהירות אם היא מופיעה בתוך T ($m \ll n$). נרצה גם לדעת איפה נמצאים כל המופעים של P בתוך T.

נוכל לעשות את זה בעזרת Suffix Tree. בזמן העיבוד המקדים, נבנה Suffix Tree בזמן לינארי (ב־n). בהינתן מחרוזת קצרה P, נחפש אותה ב־Suffix Tree. אם לא נתקענו בחיפוש בתוך ה־Suffix Tree, אז P מופיעה בתוך T. כל עלה בעץ מתחת לצומת שבו עצרנו משוייך למופע של P בתוך T. באמצעות BFS או DFS בתת־העץ, נוכל לקבל את כל k המופעים בזמן של $\mathcal{O}(n+k)$.



איור 10.1: דוגמה ל־Trie. הוא מייצג את המחרוזות aeef, ad, bbfe, bbfg ו־c.



איור 10.2: דוגמה ל־Trie דחוס, המייצג את אותו הדבר כמו ה־Trie באיור 10.1.

הגדרה 110 (Suffix Tree מוכלל): בהינתן קבוצת מחרוזות S, *Suffix Tree* מוכלל של S הוא Trie דחוס של כל הסיומות (Suffixes) של כל מחרוזת $s \in S$.
כדי לשייך כל סיומת למחרוזת שלה, נוסיף בכל סוף מחרוזת תו חדש משלה.

מה אפשר לעשות עם זה? אפשר למצוא את המופע של המחרוזת P בתוך כל המחרוזות ב־S. אפשר גם למצוא את תת־המחרוזת הארוכה ביותר של שתי מחרוזות (Longest Common Substring). כל צומת בעץ שיש לו עלה שמסיים את s_1 ועלה שמסיים את s_2 בתת־העץ שלו מייצג תת־מחרוזת משותפת ל־s_1 ו־s_2. כדי למצוא את תת־המחרוזת המשותפת הארוכה ביותר של s_1 ו־s_2, נחפש את הצומת העמוק ביותר בעץ שיש בתת־העץ שלו עלים של s_1 ושל s_2.
אם ה־Suffix Tree שלנו ידע לענות ביעילות גם על שאילתות של Lowest Common Ancetor, אז נוכל למצוא גם את ה־Longest Common Prefix של כל 2 סיומות. כמו כן, אפשר להשתמש ב־Suffix Tree למציאת פלינדרום מקסימלי במחרוזת. לכל אינדקס i במחרוזת יש פלינדרום מקסימלי באורך זוגי שמרכזו בין i ל־$i-1$. נוכל למצוא את כל הפלינדרומים במחרוזת בזמן של $\mathcal{O}(n)$.

10.3 בניית Suffix Tree בזמן לינארי

נראה כאן את האלגוריתם של Ukkonen.
נתחיל מהאות הראשונה של s, ונוסיף אותה ל־Suffix Tree (בלי שום סיומת של אות מיוחדת). את העלה שקיבלנו נסמן ב־1.
נמשיך עם האות השנייה. נוסיף אותה לעלה שיצרנו מקודם, וכן ניצור קשת חדשה מהשורש שמייצגת את האות השנייה (אם יש בכך צורך). נסמן את העלה (אם הוא נוצר) ב־2.
כשנגיע לאות ה־i (של s), נעבור על כל העלים, ונוסיף להם את האות ה־i, וניצור התפצלויות אם צריך. לבסוף, נוסיף מהשורש את האות ה־i. נמשיך כך עד שתסיים המחרוזת. את הפעולה הזאת מבצעים גם עבור התו המיוחד שמסיים את המחרוזת.
בסופו של דבר, יהיו לנו 3 כללי הרחבה לעץ:

1. הסיומת מסתיימת בעלה. במקרה הזה, אנחנו מוסיפים את האות לקשת שנכנסת לעלה.
2. הסיומת מסתיימת איפשהו באמצע קשת או בצומת שאינו עלה. במקרה הזה, אנחנו מוסיפים עלה חדש לעץ, ובמידת הצורך מפצלים קשת על ידי הוספת צומת באמצע (אם הסיומת הסתיימה באמצע הקשת).
3. הסיומת כבר קיימת, וכוללת את האות שאותה אנחנו מוסיפים. במקרה הזה, לא צריך לעשות כלום.

נשים לב לתופעה הבאה: בהרחבה הראשונה, בהכרח נסיים בעלה (כי אנחנו מרחיבים את הסיומת הכי ארוכה). זה מביא אותנו לכלל הראשון. כך יקרה גם בהרחבות הבאות, עד שנגיע לסיומת שהיא תחילית של סיומת אחרת (אם בכלל).
נניח שהגענו לסיומת כזו, ולא עצרנו בעלה. אזי גם בכל ההרחבות העתידיות (של הפאזה הנוכחית) לא נעצור בעלה, כי גם כל הסיומות הבאות הן תחיליות של סיומות אחרות שבאו לפניהן. לכן, מעתה נפעיל את הכלל השני.
נניח שהגענו לסיומת שהתו הבא בעץ הוא התו אותו אנו רוצים להוסיף. זהו הכלל השלישי. מעתה ואילך, אנחנו נמשיך עם הכלל השלישי לכל הסיומות שנשארו, כי הסיומות

הבאות שאנחנו נמצא הן סיומות של הסיומת שראינו עכשיו שמכילה את התו אותו אנו מוסיפים.

נשים לב לחוקיות שקיבלנו: בהתחלה, נצטרך לפעול לפי הכלל הראשון. בשלב כלשהו, (אולי) נגיע למצב שבו צריך לפעול לפי הכלל השני, ואז נמשיך עם הכלל השני עד ש(אולי) נגיע למצב שבו צריך להפעיל את הכלל השלישי. מהנקודה הזאת ואילך, נפעיל רק את הכלל השלישי עד סוף הפאזה. מכיוון שהכלל השלישי אינו מצריך שום פעולה, מהרגע שנתקלנו בפעם הראשונה בכלל השלישי, אפשר לסיים את הפאזה ולעבור לפאזה הבאה.

נשים לב שכשאנחנו מפעילים את הכלל הראשון, אנחנו לא באמת עושים שום דבר משמעותי מבחינת מבנה הנתונים. המבנה נשאר זהה (מבחינת העץ), אבל אנחנו רק מוסיפים תו על הקשת של ה־Trie הדחוס. למעשה, גם את זה אנחנו לא עושים, כי כל קשת מכילה מצביעים למקומות ההתחלה והסיום של תת־המחרוזת של הקשת מתוך המחרוזת המקורית. בייצוג הזה, אנחנו לא צריכים לעשות שום דבר כשמפעילים את הכלל הראשון. קשתות שמסתיימות בעלים מייצגות תתי־מחרוזות שעדיין לא נקבעה נקודת הסיום שלהן.

עוד נקודה מעניינת שכדאי לשים לב אליה היא שאם הפעלנו את הכלל השני על סיומת כלשהי בפאזה מסויימת, בפאזה הבאה, הכלל הראשון יופעל על הסיומת הזו, כי כבר הארכנו את הסיומת לכדי סיומת חדשה, שאין שום סיומת יותר ארוכה ממנה שמתחילה בצורה הזאת. מכיוון שהפעלה של הכלל הראשון אינה דורשת פעולה בייצוג שלנו, נוכל להתחיל את הפאזה הבאה מהנקודה הראשונה שבה נתקלנו בכלל השלישי בפאזה הנוכחית.

נסביר בצורה טובה יותר: מכיוון שהפעלה של הכלל הראשון והשלישי לא דורשות פעולה בייצוג הנוכחי, אין שום סיבה להתעכב על הפעלתם. הכלל השני הוא היחיד שדורש פעולה. הכללים יופיעו לפי הסדר (כלומר ראשון, שני ואז שלישי). כמו כן, סיומת שהופעל עליה הכלל השלישי בפאזה ה־i, הכלל הראשון יופעל עליה בפאזה ה־$i+1$. לכן, נוכל להתחיל את הפאזה ה־$i+1$ בסיומת הראשונה שבה נתקלנו בכלל השלישי בפאזה ה־i (כי עד אז לא נדרשת פעולה, נתקל רק בכלל הראשון).

נשים לב שעם כל הייעולים היפים שראינו כאן, הסיבוכיות של האלגוריתם היא עדיין $\mathcal{O}\left(n^2\right)$, כי בכל פעם שעוברים לסיומת הבאה, אנחנו צריכים להתחיל לחשב את הסיומת מהתחלה, וזה לוקח $\mathcal{O}\left(n\right)$ זמן.

Suffix Links 10.4

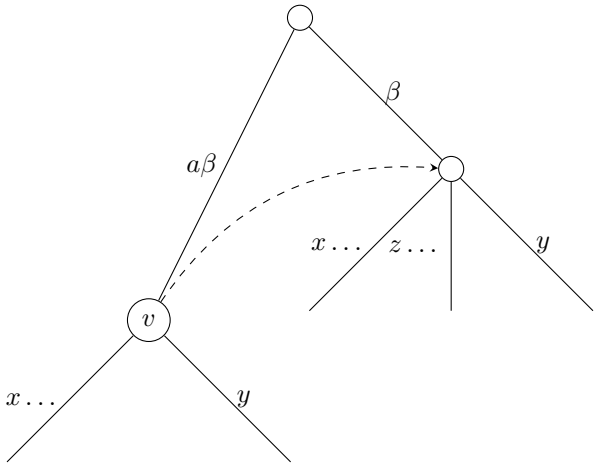
הגדרה 111 (Suffix Link): נגדיר *Suffix Link* להיות מצביע מהצומת שמייצג את המחרוזת $a\beta$ (כאשר a היא אות ו־β היא מחרוזת) ב־Suffix Tree לצומת שמייצג את המחרוזת β ב־Suffix Tree. כל זה בהנחה שקיימים הצמתים המייצגים את המחרוזות $a\beta$ ו־β.

ניתן למצוא דוגמה ל־Suffix Link באיור 10.3.

האם באלגוריתם של Ukkenon יש צמתים כאלה? נשתמש בסימונים של איור 10.3. נניח שהצומת v נוצר בהפעלה של הכלל השני. אזי הייתה סיומת $a\beta x\ldots$, והוספנו את הסיומת $a\beta y$. לכן, הייתה גם סיומת $\beta x\ldots$. אם הייתה גם סיומת $\beta z\ldots$, אזי הצומת של β כבר היה קיים, ולכן אפשר לייצר את ה־Suffix Link. אחרת, כשנסרוק באיטרציה הבאה בפאזה את הסיומת βy, נפעיל את הכלל השני, ואז נייצר את הצומת של β.

נשים לב לאינווריאנטה הבאה: כל ה־Suffix Link־ים כבר קיימים, פרט אולי לצמתים האחרונים שהוספנו זה עתה (בהפעלה של הכלל השני).

איך נייצר Suffix Link בצורה יעילה אם הוא לא קיים בצומת הנוכחי? נניח שאנחנו נמצאים עכשיו בצומת של המחרוזת $a\beta\delta$, ואנחנו רוצים ליצור Suffix Link לצומת של המחרוזת $\beta\delta$. נעשה את זה כך:



איור 10.3: דוגמה ל־Suffix Link (הקו המקווקו).

1. נעלה לצומת האבא של הצומת הנוכחי (זה הצומת של המחרוזת $a\beta$). מהאינווריאנטה, כבר יש בו Suffix Link.
2. נלך ב־Suffix Link שלו ונגיע לצומת של המחרוזת β.
3. נרד למטה בעץ לפי המחרוזת δ. אולי כאן אנחנו צריכים להפעיל את הכלל השני (לסיומת $\beta\delta$) ולפצל קשת באמצע. עכשיו אנחנו נמצאים בצומת שמייצג את המחרוזת $\beta\delta$, ואליו צריך להגיע ה־Suffix Link.

10.5 ניתוח

הפעלות של הכללים הראשון והשלישי מבוצעות בזמן קבוע, ולכן בסך הכל הכללים הראשון והשלישי דורשים $\mathcal{O}(n)$ זמן להפעלה.

את הכלל השני אנחנו מפעילים $\mathcal{O}(n)$ פעמים. לרוע המזל, אי אפשר לומר שכל הפעלה שלו מבוצעת בזמן קבוע. עלייה לצומת האבא מבוצעת בזמן קבוע, אולם ירידה חזרה לפי המחרוזת δ עשוייה להכיל יותר ממעבר על קשת אחת.

נעקוב אחרי העומק (בקשתות) של הצומת הנוכחי. כשמפעילים את הכלל הראשון, העומק גדל בלכל היותר 1. כשעוקבים אחרי ה־Suffix Link, העומק יכול לקטון בלכל היותר 1. בתחילת האלגוריתם, העומק שלנו הוא 0. בכל שלב של האלגוריתם, העומק המירבי חסום על ידי n (כי זה מספר האותיות שיש במחרוזת). בכל שאר המקרים, העומק לא משתנה. מאחר והעומק של הצומת הנוכחי מתחיל ב־0 ותמיד קטן מ־n, הוא יכול לגדול לכל היותר $3\cdot n$ פעמים, כי יש לכל היותר $2\cdot n$ הקטנות. מספר הפעמים שנעים על קשת הוא לכל היותר $5\cdot n$. בכל מעבר על קשת עושים עבודה קבועה.

משפט 112 (זמן הריצה של Ukkonen): זמן הריצה של האלגוריתם של *Ukkonen* למציאת *Suffix Tree* הוא לינארי ב־n – $\mathcal{O}(n)$.

10.6 Suffix Arrays

הבעיה העיקרית ב־Suffix Trees היא הכמות הגדולה של הזיכרון הדרושה לתחזור מבנה הנתונים (בקבועים, לא אסימפטוטית). אם אנחנו באמת רוצים מעבר על קשת בזמן קבוע, אנחנו צריכים מערך של מצביעים בגודל $|\Sigma|$ (גודל האלפבית) בכל צומת. זה המון!

עם Suffix Arrays אנחנו מקבלים קבועים הרבה יותר קטנים, אבל יחד עם זאת, הם מוגבלים יותר במה שהם מאפשרים.

איך זה עובד? ה־Suffix Array נותן לנו מערך של אינדקסים בתוך המחרוזת הגדולה. כל אינדקס הוא התחלה של Suffix במחרוזת, וה־Suffix־ים מסודרים לפי סדר לקסיקוגרפי.

אפשר לבנות Suffix Array בזמן לינארי: בונים Suffix Tree (בזמן $\mathcal{O}(n)$), ואז עוברים על העץ בשיטת DFS לפי סדר לקסיקוגרפי, וכך ממלאים את ה־Suffix Array (גם ב־$\mathcal{O}(n)$).

איך אפשר לחפש תת־מחרוזת בעזרת Suffix Arrays? אם P מופיעה ב־T, אז כל המופעים שלה יופיעו ברצף ב־Suffix Array. נבצע חיפוש בינארי על ה־Suffix Array, ונקבל סיבוכיות של $\mathcal{O}(m \cdot \log n)$.