ביג דאטה – פתרון תרגיל 3

הקוד הרלוונטי לשאלות התכנותיות, נכתב במטלב והוא מצורף כנספח.

1.

a. נסמן $C_{dxd} = A^tA - B^tB$ אזי למעשה נרצה להוכיח את נכונות המשפט:

$$||C||_2^2 = \lambda_1^2 - the\ spectral\ norm$$

נוכיח תחילה כי C סימטרית:

$$C^t = (A^tA - B^tB)^t = (A^tA)^t - (B^tB)^t = A^tA - B^tB = C$$

האמת שקל לראות את זה כיוון ש-A^tA ו-B^tB הן סימטריות.

כעת כיוון שכל מטריצה סימטרית ממשית היא בעלת ערכים עצמיים ממשיים וניתנת ללכסון אורתוגונלי, ניתן להציגה לפי הפירוק לערכים עצמיים:

$$C = V\Sigma' V^t$$

כיוון שהמטריצה Σ בפירוק SVD נקבעת ביחידות לכל מטריצה. נקבל שהערכים הסינגולריים של C הם בדיוק הערכים העצמיים (זה גם די טבעי ויש משפט שמתייחס לכך).

C^tC היא הרמיטית ולכן הערכים העצמיים שלה ממשיים. כמו כן, היא positive semi-definite כיוון שמתקיים $\forall x: x^tC^tCx = (Cx)^t(Cx) \geq 0$, מכאן שהערכים העצמיים שלה הם אינם שליליים.

נסמן את הערכים העצמים של C^tC כך:

$$\lambda_1^2 \geq \lambda_2^2 \geq \cdots \geq \lambda_d^2 \geq 0$$

ובהתאם לכך שאלו ריבועי הערכים הסינגולריים, נקבל כי במקרה זה הם בדיוק ריבועי הערכים העצמיים.

בהתאם לסימון זה, וכפי שראינו בשיעור, הוקטורים של V יהיו אם כן הוקטורים העצמיים של C^tC המתאימים לערכים העצמיים שהגדרנו:

$$\forall i = 1, \dots, d: (C^tC)V_i = \left(\lambda_i^2\right)V_i$$

וקטורים אלו הם כמובן בסיס אורתונורמלי למרחב ועל כן נוכל לייצג באמצעותם כל וקטור.

אם כן, יהי $x = \sum_{i=1}^{d} \alpha_i V_i$, אזי נקבל:

$$(C^tC)x = \sum_{i=1}^{d} \alpha_i \lambda_i^2 V_i$$

ולבסוף נקבל:

$$||Cx||_2^2 = (Cx)^t(Cx) = x^t(C^tC)x = \left(\sum_{i=1}^{d} \alpha_i^t V_i^t\right)\left(\sum_{i=1}^{d} \alpha_i \lambda_i^2 V_i\right) = \sum_{i=1}^{d} |\alpha_i|^2 \lambda_i^2$$

כאשר המעבר האחרון נובע מכך שהוקטורים של V הם כאמור אורתונורמלים.

כעת:

$$||Cx||_2^2 = \sum_{i=1}^{d} |\alpha_i|^2 \lambda_i^2 \leq \lambda_1^2 \sum_{i=1}^{d} |\alpha_i|^2 = \lambda_1^2 ||x||_2^2$$

כל שנותר בשלב זה הוא למצוא וקטור המקיים $||y|| = 1$ עבורו השוויון מתקיים. אך קל לראות כי V_1 אשר לו מתאים הערך העצמי λ_1^2 מקיים את השוויון.

למעשה עד כה הוכחנו כי:

$$||C||_2^2 = \lambda_1^2$$

b. נסמן ב-$r = rank(A)$,$SVD(A) = U\Sigma V^t$ אזי תהי המטריצה $B_{lxd} = \Sigma' V'^t$ כאשר Σ' מורכבת מ-l השורות והעמודות הראשונות ב-Σ, ו-V'^t מורכבת מ-l השורות הראשונות ב-V^t.
נוכיח כי מתוך כל המטריצות B ,lxd ממזערת את הביטוי:

$$\left||A^tA - B^tB|\right|_2^2$$

בסימון שלנו, ובסימון λ_i מהסעיף הקודם נקבל:

$$\left||V\Sigma^2V^t - V'\Sigma'^2V'^t|\right|_2^2 = \left\|\sum_{i=1}^{r}\lambda_i^2V_iV_i^t - \sum_{i=1}^{l}\lambda_i^2V_iV_i^t\right\|_2^2 = \left\|\sum_{i=l+1}^{r}\lambda_i^2V_iV_i^t\right\|_2^2 = \lambda_{l+1}^2$$

נסמן $A_k = \sum_{i=1}^{k}\lambda_iU_iV_i^t$ יהי $w \in R^d$, אזי בהינתן ש-$l < r$ וכן $l < d$ נקבל כי:

$$span\big(rows(A_{l+1})\big) = span(V_1^t, \dots, V_{l+1}^t) \subseteq span\big(rows(A^tA)\big)$$
$$span(w_1, \dots, w_{d-l}) \cap span(V_1^t, \dots, V_{l+1}^t) \neq \emptyset$$

כלומר קיים $w \in R^d$ כך ש:

$$B^tBw = 0 \wedge A^tAw = A_{l+1}^tA_{l+1}w$$

בהצבה חזרה נקבל:

$$\left|\left|(A^tA - B^tB)w\right|\right|^2 = \left||A^tAw|\right|^2 = \sum_{i=1}^{l+1}\lambda_i^2\left||V_i^tw|\right| = \lambda_{l+1}^2$$

כאשר השוויון האחרון נובע מכך ש-$B^tBw = 0$. למעשה מצאנו מקרה בו מתקיים השוויון.
כעת כיוון שמסעיף a, ידוע לנו כי $\left||A^tA - B^tB|\right|_2^2$ הוא הערך העצמי הכי גדול של $A^tA - B^tB$, נקבל ש-B כפי שהוגדרה בתחילת סעיף זה, מביאה אותנו לפתרון מינימלי עבור כל המטריצות מסדר lxd.

c. האלגוריתם כפי שהוצג בתרגיל (כפי שהוצג בכיתה למעשה), שונה מזהו אשר מציג ליברטי במאמר. ליברטי במאמר בכל שלב i למעשה הקטין את כל אחד מהערכים הסינגולרים ל-$\max\left(o, \sqrt{s_i - s_{\frac{l}{2}}}\right)$, כלומר, בכל שלב לא אופסה רק השורה ה-l של B^i אלא אופסו לפחות חצי מהשורות. היתרון בכך היה שאין צורך לחשב SVD בכל שלב אלא רק כל $\frac{l}{2}$ שלבים לפחות.
המחיר שהאלגוריתם משלם הוא פי 2 בקירוב. אם כן, אנו נוכיח כי האלגוריתם שהוצג בתרגיל (וכאמור בכיתה) מגיע ל-$\frac{||A||_F^2}{l}$ ולא $\frac{2||A||_F^2}{l}$, כפי שפורסם במאמר. נסמן B_+ באיטרציה ה-i, כך $B_+ = U^i\Sigma^i\left(V^i\right)^t$, וכן $B^i = \Sigma'^i\left(V^i\right)^t$ על מנת להקל על הסימון בהוכחה.
נציג את הצעדים המרכזיים בהוכחה בהתבסס על ההוכחה במאמר עצמו:
על מנת להתאים למינוח במאמר, נגדיר $C^i = \Sigma^i\left(V^i\right)^t$ בשלב ה-i. נשים לב שבסימון זה, כיוון שהשורה ה-l של B^{i-1} תמיד מאופסת בתחילת השלב ה-i, מתקיים:

$$\forall x: \langle A_i, x\rangle^2 + \left|\left|B^{i-1}x\right|\right|^2 = \left||C^ix|\right|^2$$

חשוב לשים לב כי A_i הוא וקטור השורה ה-i ועל כן ה-notation. קל לראות את זה אם מסתכלים על העובדה ש-A_i מושם לשורה המאופסת ב-B^{i-1} ומהפירוק של המטריצה המתקבלת ל-SVD מרכיבים את C^i. כיוון שהוא מתקבל על ידי הכפלה משמאל ב-$\left(U^i\right)^t$, הוא למעשה מתקבל על ידי שינוי בסיס (כי U אוניטארית), על כן, ה"גודל" שהוא משרה זהה ל"גודל" שמשרה המטריצה המקורית.
לכן, נשים לב כי:

$$0 \le \left|\left|\hat{B}^t\hat{B}\right|\right| \le ||A^tA||$$

הנ"ל נכון כיוון ש-P^tP היא מטריצה Positive semi-definite לכל מטריצה P, וכן כיוון ש-$\hat{B}$ מתקבלת מהשורות של A ובכל איטרציה יכולות להיות מכווצות לפי הערך הסינגולרי הקטן ביותר:

$$||Ax||_2^2 - \left||\hat{B}x|\right|_2^2 = \sum_{i=1}^{n}\langle A_i, x\rangle^2 + \left||B^{i-1}x|\right|_2^2 - \left||B^ix|\right|_2^2 = \sum_{i=1}^{n}\left||C^ix|\right|_2^2 - \left||B^ix|\right|_2^2 \ge 0$$

כאשר המעבר האחרון טריוויאלי מההגדרה של B^i הבנויה מכיווץ של Σ אשר מגדירה את C^i.
כעת, נעבור להוכחת החסם עצמו, יהי x הוקטור העצמי המתאים לערך העצמי הגדול ביותר של $A^tA - B^tB$, אזי:

$$\begin{aligned}
\left||A^tA - \hat{B}^t\hat{B}|\right|_2 &\underset{section\ a}{=} x^t\left(A^tA - \hat{B}^t\hat{B}\right)x = ||Ax||_2^2 - \left||\hat{B}x|\right|_2^2 \\
&= \sum_{i=1}^{n}\langle A_i, x\rangle^2 + \left||B^{i-1}x|\right|_2^2 - \left||B^ix|\right|_2^2 = \sum_{i=1}^{n}\left||C^ix|\right|_2^2 - \left||B^ix|\right|_2^2 \\
&= \sum_{i=1}^{n} x^t\left(C^i\right)^tC^ix - x^t\left(B^i\right)^tB^ix = \sum_{i=1}^{n} x^t\left(\left(C^i\right)^tC^i - \left(B^i\right)^tB^i\right)x \\
&= \underset{by\ definition\ of\ norm\ 2}{\le} \sum_{i=1}^{n}\left|\left|\left(C^i\right)^tC^i - \left(B^i\right)^tB^i\right|\right|_2^2
\end{aligned}$$

כעת נציב חזרה את ההגדרות של $B^i = \Sigma' V^t$ ו-$C^i = \Sigma V^t$ ונקבל:

$$||Ax||_2^2 - \left||\hat{B}x|\right|_2^2 \le \sum_{i=1}^{n}\left|\left|\left(C^i\right)^tC^i - \left(B^i\right)^tB^i\right|\right|_2^2 = \sum_{i=1}^{n}\left|\left|\left(\Sigma^i\right)^2 - \left(\Sigma'^i\right)^2\right|\right|_2^2 = \sum_{i=1}^{n}\delta_i \quad (*)$$

בשלב זה כל שנותר הוא לחסום את $\sum_{i=1}^{n}\delta_i^2$:, ופה באה לעזרתנו נורמת פרובניוס:
גם כאן נשים לב כי מתקיים:

$$||A_i||_2^2 = \left||C^i|\right|_F^2 - \left||B^{i-1}|\right|_F^2$$

כאשר עבור וקטורים, למשל A_i, נורמת פרובניוס זהה לנורמה L_2. השיקולים הם זהים במקרה זה – הנורמה נקבעת לפי "גודל" המטריצות ולא הבסיס בו הן מוצגות ולכן C^i שהיא סיבוב של החיבור של השתיים האחרות משרה את אותו ה"גודל" כמו האחרות.

$$\begin{aligned}
\left||\hat{B}|\right|_F^2 &\underset{by\ definition\ of\ \hat{B}}{=} ||B^n||_F^2 = ||B^n||_F^2 - 0 = ||B^n||_F^2 - ||B^0||_F^2 \\
&= \sum_{i=1}^{n}\left||B^i|\right|_F^2 - \left||B^{i-1}|\right|_F^2 \\
&= \sum_{i=1}^{n}\left(\left||C^i|\right|_F^2 - \left||B^{i-1}|\right|_F^2\right) - \left(\left||C^i|\right|_F^2 - \left||B^i|\right|_F^2\right) \\
&= \sum_{i=1}^{n}||A_i||_F^2 - \sum_{i(=1)}^{n} tr\left(\left(C^i\right)^tC^i - \left(B^i\right)^tB^i\right) \\
&= ||A||_F^2 - \sum_{i(=1)}^{n} tr\left(\left(\Sigma^i\right)^2 - \left(\Sigma'^i\right)^2\right) = ||A||_F^2 - \sum_{i=1}^{n} l\cdot\delta_i
\end{aligned}$$

מצד שני, לפי ההגדרה:

$$\left|\left|\hat{B}\right|\right|_F^2 \geq 0$$

ומשילוב של השניים נקבל:

$$\frac{||A||_F^2}{l} \geq \sum_{i=1}^{n} \delta_i$$

כעת, מהצבה חזרה ב-$(*)$:

$$\left|\left|A^t A - \hat{B}^t \hat{B}\right|\right|_2 \leq \sum_{i=1}^{n} \delta_i \leq \frac{||A||_F^2}{l}$$

וזהו החסם העליון, האמת שבכל מקרה סביר $\left|\left|\hat{B}\right|\right|_F^2 > 0$ ולכן החסם בצורתו זו לא יתקבל אלא במקרה מאוד רדוד.

2. על מנת להוכיח כי האלגוריתם סופר כל משולש סגור רק פעם אחת נתבונן באפשרויות ליצור משולש פתוח. על פי ההגדרה, האלגוריתם בונה משולש פתוח (y,x,z) אם ורק אם $x \prec y \; and \; y \prec z$ וכן $Friends(\{x,y\}), Friends(\{x,z\})$ וסופר אותו רק במידה ו-$Friends(\{y,z\})$ (הסימון הנ"ל יקרא להלן סימון ההגדרה). אזי יהיו הקודקודים i,j,k ובלי הגבלת הכלליות נניח כי $i \prec j \prec k$ וכמובן נניח ש-$Friends(\{i,j\}), Friends(\{i,k\})$. משלושת הקודקודים הללו ניתן לבנות את המשולשים הפתוחים הבאים:
   - (i,j,k) – משולש זה לא ייבנה כיוון ש-$i \prec j$, וזה סותר את התנאי ש-$x \prec y$ בסימון ההגדרה (כמו כן ייתכן כי בכלל לא מתקיים $Friends(\{j,k\})$).
   - (k,j,i) – משולש זה לא ייבנה כיוון ש- $i \prec j$ וכן $j \prec k \Rightarrow i \prec k$ ועל כן סותר גם את $x \prec y$ וגם את $y \prec z$ בסימון ההגדרה (כמו כן ייתכן כי בכלל לא מתקיים $Friends(\{j,k\})$).
   - (i,k,j) – משולש זה לא ייבנה כיוון ש-$j \prec k \Rightarrow i \prec k$ ולכן המשולש סותר את התנאי $x \prec y$ (כמו כן ייתכן כי בכלל לא מתקיים $Friends(\{j,k\})$).
   - (j,k,i) – משולש זה לא ייבנה כיוון ש-$j \prec k$ ולכן המשולש סותר את התנאי $x \prec y$ (כמו כן ייתכן כי בכלל לא מתקיים $Friends(\{j,k\})$).
   - (k,i,j) – משולש זה לא ייבנה כיוון ש-$j \prec k$ ולכן המשולש סותר את התנאי $y \prec z$.
   - (j,i,k) – משולש זה ייבנה – זאת כיוון ש-$i \prec j \; and \; j \prec k$ וכן $Friends(\{i,j\}), Friends(\{i,k\})$ – אם אחד מאלו לא היה מתקיים לא נוצר המשולש כאמור.

כלומר מתוך כלל האפשרויות לבנות את משולש פתוח מהקודקודים $3! = 6$, רק אחד ייבנה, על כן לכל שלשת קודקודים, לא יווצר יותר ממשולש פתוח אחד, ומכאן שבוודאי לא ייספר יותר מאחד. אם כן נותר להוכיח כי במידה וזהו משולש סגור, הוא אכן ייספר:

כעת, בשלב 2 האלגוריתם יבדוק האם $Friends(\{j,k\})$, במידה וכן, הוא יספור את המשולש ובכל מקרה, ימשיך הלאה למשולש הפתוח הבא, כנדרש.

על כן, מכל שלשת קודקודים יווצר לכל היותר משולש פתוח אחד שיספר לכל היותר פעם אחת, והוא אכן יווצר ויספר במידת הצורך – כלומר האלגוריתם סופר כל משולש סגור פעם אחת.

3. תחילה נתאר את הרעיון המסדר מאחורי האלגוריתם:
עיקר האלגוריתם בנוי על כך שמספיק לדעת כי בין שני קודקודים, $y \prec z$, קיימת צלע, $Friends(\{y,z\})$, על מנת לדעת שהם סוגרים את כל המשולשים הפתוחים שנוצרו על ידי קודקודים $x \prec y$ כך ש-$Friends(\{x,z\}), Friends(\{x,y\})$. כלומר, אם נרכז כ-*value* לכל זוג כזה את כל המשולשים הפתוחים

שצריכים לדעת אם הצלע קיימת, ניתן יהיה באמצעות $|value|$ לדעת במיידי את כמות המשולשים שהצלע סוגרת, וכך לסכום בקלות לכל המשולשים.
הדרך שבה נעשה את זה תהיה תלת-שלבית:

1. תחילה, ל-map נקבל כקלט צלע - זוג קודקודים, x, y כך ש-$Friends(\{x, y\})$ (מן הסתם תחת ההנחה שאנחנו שומרים רק צלעות – ניתן היה באופן דומה לעבוד עם שמירה של קודקוד וכל הקודקודים שמקושרים אליו ובשלב זה רק להוציא זוגות שלו עם קודקודים שגדולים ממנו). אם ה-id של הראשון קטן יותר, ה-map יוציא אותו כמפתח ואת הקודקוד השני בתור value (עושים זאת רק בתנאי שה-ID הראשון קטן על מנת לא להוציא כל צלע פעמיים):

MAP1($tuple\ a = \langle x, y \rangle$)
1. If $x \prec y$ then:
   1.1. EmitIntermediate(x, y)
2. end

ה-reduce במקרה זה, יקבל כמפתח קודקוד x וכ-value את כל הקודקודים, y, המקיימים $x \prec y$ וכן $Friends(\{x, y\})$. הפונקציה עצמה תייצא אם כן את כל זוגות הקודקודים, $\langle y, z \rangle$, המהווים משולש פתוח מהצורה (y, x, z) (אשר יהווה מפתח ב-map הבא), וכן ערך לא משמעותי אך לצורך העניין יהיה x (וב-map ישמש כ-value שלו). כמו כן, כל זוג $\langle x, y \rangle$ עם איזשהו ערך ייחודי, "closed", אשר יזהה לנו כי צלע זו קיימת (כך נדע אם המשולשים נסגרים או לא).

REDUCE1($String\ ID = x, List\ Neighbors$)
1. For y in $Neighbors$ do:
   1.1. Emit($\langle x, y \rangle$, "$closed$")
   1.2. For z in $Neighbors$ do:
      1.2.1. If $y \prec z$ then:
         1.2.1.1. Emit($\langle y, z \rangle$,x)
      1.2.2.end
   1.3. end
2. end

2. בשלב זה נקבל כקלט זוג קודקודים וערך, במקרה זה, קודקוד אשר עבורו הם מהווים משולש פתוח. כמו כן, ישנו ערך "closed" לכל צלע קיימת – כלומר, ערך שמזהה כי הצלע קיימת ועל כן המשולשים בעצם סגורים. לכן בשלב זה, ה-map לא יעשה דבר (יוציא את הפרמטרים כמו שנכנסו), אלא ייתן ל-sort שלפני ה-reduce לעשות את שלו:

MAP2($tuple\ key = \langle x, y \rangle, value$)
1. EmitIntermediate($\langle x, y \rangle, value$)

הטריק הוא שבאמצעות ה-sort ה-value הופך ל-List של כלל הקודקודים שעבורם הזוג הוא משולש פתוח וכן במידה וקיים המזהה הייחודי "closed". כלומר, אם ערך זה קיים, גודל הקבוצה הוא מספר המשולשים הפתוחים שצלע זו סוגרת (כיוון שהיא קיימת), ועל כן נוציא את $|value| - 1$ (פחות הערך "closed" עצמו):

REDUCE2($tuple\ key = \langle x, y \rangle, List\ triangles$)
1. If ("$closed$" in $triangles$) do:
   1.1. Emit($|triangles|$-1)

3. כעת כל שנותר לנו לקבל ב-map האחרון את כמויות המשולשים ולמפות אותם לאותו הערך, כך כשיעברו sort לקראת ה-reduce כל כמויות המשולשים ימופו מפתח ונסכום אותם:

MAP3($Integer\ numOfTriangles$)

1. EmitIntermediate($0, numOfTriangles$)

REDUCE3($Integer\ key = 0, List\ numsOfTriangles$)

1. count ← 0
2. For $numOfTriangles$ in $numsOfTriangles$ do:
   2.1. count ←count + $numOfTriangles$
3. end
4. Emit(count)

כך נקבל את מספר המשולשים הסגורים בגרף.

4.

a. על מנת שהאלגוריתם הסדרתי כפי שהוגדר יוכל לרוץ על הקלט, נניח כי כל קודקוד נשמר יחד עם קבוצת השכנים שלו (פונקצית ה-*MAP1* המתאימה למקרה זה היא כאמור הוצאה של כל זוג של הקודקוד עם קודקודים הגדולים ממנו). תחת הנחה זו, האלגוריתם הסדרתי בצורתו היעילה ירוץ לכל קודקוד על רשימת השכנים שלו, ויבדוק לכל זוג שכנים גדולים ממנו האם יש ביניהם צלע ואם כן יספור (כמובן לא פעמיים) – כלומר לא ישמור את הרשימה של משולשים פתוחים אלא רק יבדוק אם המשולש סגור.
עם זאת, האלגוריתם המתואר בשאלה אכן יוצר רשימה של כלל המשולשים, על כן מבחינת סיבוכיות זמן ריצה:
יהיה n מספר הקודקודים, אזי רצים לכל קודקוד על כל הקודקודים השכנים שלו, ולכל שכן שוב רצים על כל השכנים (כאשר ישנם לכל היותר $n-1$ שכנים) ובשלב השני עוברים על מספר המשולשים הפתוחים שנוצרו לכן סך הכל (תחת ההנחה שיצירה של משולש, למעשה tuple של מצביעים לקודקודים לוקחת $O(1)$):

$$O(n^3 + P2)$$

(ניתן היה ליעל ל-$O(n^2 + P2)$ אם היינו שומרים לכל קודקוד רק את הגדולים ממנו)
מבחינת סיבוכיות זכרון:
עלינו לזכור לכל קודקוד את השכנים שלו (מלכתחילה n – קודקודים אשר לכל אחד $O(n)$ שכנים) וכן את כלל המשולשים הפתוחים (כאשר כל משולש מורכב משלושה מצביעים ולכן $O(1)$):

$$O(n^2 + P2)$$

בשלב זה הנחנו שנוצרת רשימת המשולשים הפתוחים במלואה לפני שעוברים עליה, ניתן היה לעבור על כל רשימה שנוצרה לכל קודקוד בנפרד ואז הזכרון היה:

$$O(n^2 + M)$$

וכאמור ניתן היה לא לשמור אף משולש ולעשות את הבדיקה תוך כדי והזכרון היה נשמר ב-$O(n^2)$.
מכאן שהפרמטרים M ו-$P2$ מאוד משפיעים על האלגוריתם הסדרתי. על זמן הריצה קל לראות שתמיד תהיה השפעה של $P2$ הרי עוברים על כלל המשולשים הפתוחים לפחות. M עצמו משפיע כאן רק כי הוא רלוונטי לכמות המשולשים הפתוחים ולכן רק דרך $P2$. הזכרון במקרה זה מושפע בהתאם למימוש, אם נבחר לרוץ על הרשימה המלאה של משולשים, $P2$ יהיה הפקטור המכריע, אם נבחר לרוץ לכל קודקוד בנפרד על רשימת המשולשים הפתוחים שיצר M יהיה הגורם המכריע, ואילו אם נחליט שכל משולש נבדק מידית, שניהם לא ישפיעו על סיבוכיות הזכרון.

עבור האלגוריתם MapReduce נעבוד תחת אותה הנחת שמירת המידע - כל קודקוד נשמר יחד עם השכנים. נניח W מספר ה-workers, כמובן שהוא יכול להשתנות בכל שלב אך לצרכי ניתוח יהיה זה יותר קל להניח שהוא קבוע, יתר על כן, אם מדובר על אותם סדרי גודל של workerים, ההשפעה על הסיבוכיות תהיה מצומצמת יחסית. חשוב לציין, לא ננסה לנתח את זמן המיון כי הוא מאוד תלוי מימוש.

אם כן:

MAP1 יוציא לכל קודקוד את כל הקודקודים הגדולים ממנו - $O\left(\frac{n^2}{W}\right)$, סיבוכיות המקום תהיה לכן $O(n^2)$. REDUCE1 לאחר מכן יעבור לכל קודקוד על כל הקודקודים גדולים ממנו - $O(M)$, זאת, תחת ההנחה המקלה שהמידע הרלוונטי לכל קודקוד עובר עיבוד ב-worker אחר. חשוב לציין כי מדובר על $O(M)$ ולא $O(n)$ כי אנו עוברים על כל הצלעות הגדולות מהקודקוד ואלו תמיד מסדר הגודל של מספר המשולשים הפתוחים שהוא יכול ליצור ולא סך הקודקודים הקיימים. במקרה זה סיבוכיות המקום תהיה $O(P2)$.

MAP2 פשוט מייצא את הנתונים באותו הצורה ולכן מדובר על סיבוכיות זמן ריצה של $O(\frac{P2}{W})$ – המקביל לכמות אותה אנו נדרשים לקרוא וסיבוכיות הפלט זהה כמובן. REDUCE2 בודק האם איבר קיים ברשימה - ניתן לבצע אפילו ב-$O(1)$ באמצעות hash, אך הביצוע הוא לכל זוג צלעות שיכולות לסגור משולש פתוח ולכן $O(\frac{\min(P2,n^2)}{W})$ וסיבוכיות המקום תהיה בהתאם: $O(\min(P2,n^2))$.

MAP3 פשוט עובר על רשימה זו ומעביר הכל לרשומה אחת ולכן, נדרש זמן ריצה של $O\left(\frac{\min(P2,n^2)}{W}\right)$, וסיבוכיות המקום של הפלט תהיה פשוט רשימה עם ערך לכל משולש סגור $O(\min(P2,n^3))$. REDUCE3 יעבור על הרשימה ויחבר – יבוצע ב-worker יחיד ולכן סיבוכיות זמן ריצה של $O(\min(P2,n^3))$ ומן הסתם סיבוכיות הפלט $O(1)$ זה counter ולכן זניח.

ולכן בסך הכל:

סיבוכיות זמן הריצה:

$O\left(\min(P2,n^3)+\frac{n^2}{W}\right)$ כיוון ש-$\frac{\min(P2,n^2)}{W}, \frac{P2}{W} < \min(P2,n^3)$ וכמובן כי $M \le P2$.

וסיבוכיות המקום תהיה לכן:

$O(n^2+P2)$ אשר נובע מהאיטרציה הראשונה, השאר מאותו סדר גודל או קטנים יותר.

גם כאן ניתן לראות כי הפרמטר $P2$ הוא המכריע, אך הפעם ההשפעה שלו על סיבוכיות המקום תמיד תהיה. כפי שציינתי, התעלמנו מנושא המיון בניתוח, עם זאת, יש לציין כי השפעת סיבוכיות המיון הוא רלוונטית שכן מדובר על 2 איטרציות עם סיבוכיות זיכרון גדולה – המיון בשלישי לא קיים.

b. הדוגמא הטובה ביותר למשפחה בה הסדר ישפיע הכי הרבה על M ו-$P2$ היא כמובן משפחת גרפי הכוכב.

נדגים:

יהי קודקוד x בגרף G המכיל n קודקודים ותהי E קבוצת הצלעות של G, כך ש:

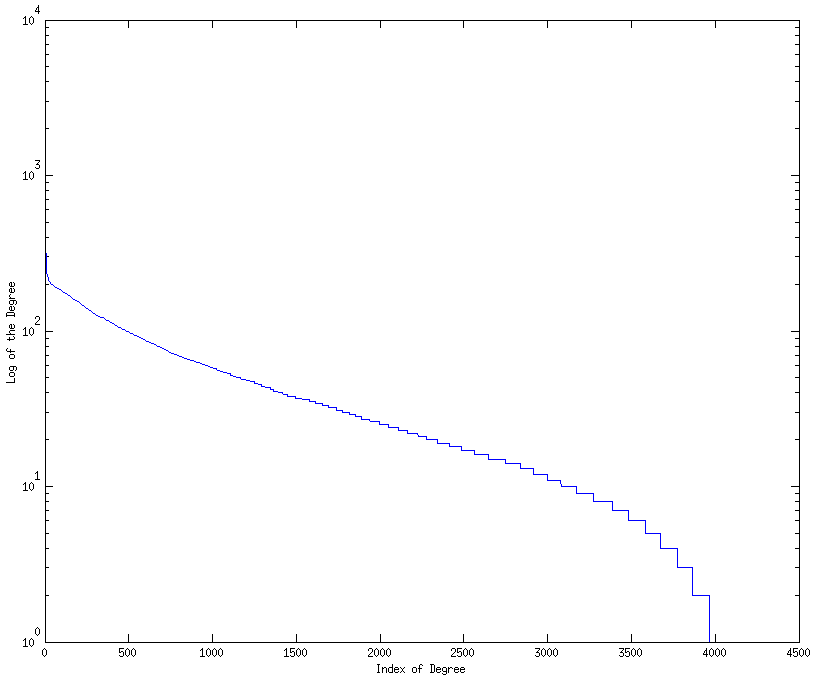
$$\forall x \neq y,z \in G: (x,y),(x,z) \in E \wedge (y,z) \notin E$$

כלומר כל הקודקודים מקושרים ל-x בלבד.

כעת, אילו נשרה סדר על הגרף בו x יהיה הקודקוד הראשון, נקבל כי $M = P2 = n(n-1)$ (n בחר 2). מאידך, לו היינו משרים סדר בו x היה הקודקוד הגדול ביותר – n, היינו מקבלים $M = P2 = 0$.

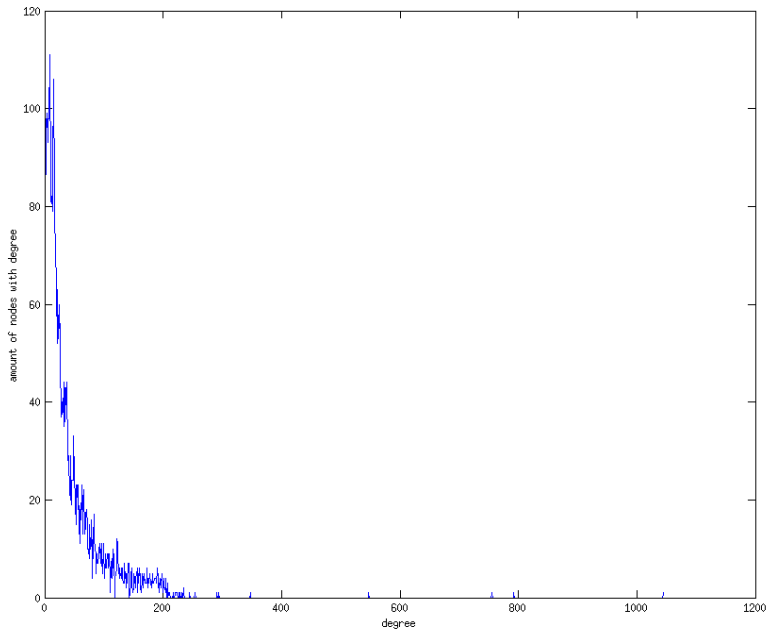
כלומר, לגרף בעל n קודקודים ההפרש יכול להגיע אף לסדר גודל של n^2.

.5



כצפוי, מעט מאוד מהאנשים מחזיקים בכמות עצומה של חברים – רק 4 אנשים מחזיקים מעל ל-10% מכמות המשתמשים, ואילו מרבית המשתמשים מחזיקים בין 10 ל-100 חברים, כאלף מחזיקים מתחת ל-10.

הגרף שאולי יותר מעניין יהיה להסתכל עליו הוא דווקא התפלגות הדרגות – כמות הקודקודים בעלי דרגה X כנגד הדרגה:



כאן למעשה מתגלה הסוד וניתן לראות כי מדובר על התפלגות הזנב הארוך (long tail).

6. לכל קודקוד $x \in G$, תהי $BigN(x)$ קבוצת השכנים של x אשר גדולים ממנו. באופן פורמלי:
$$\forall y \in G: y \in BigN(x) \Leftrightarrow x \prec y$$
נסמן $N(x) = |BigN(x)|$, נשים לב כי $P(x)$ – מספר המשולשים הפתוחים ש-x יכול לייצור, הוא בדיוק:
$$P(x) = (N(x)-1) + (N(x)-2) + \cdots + 1 = \frac{(N(x)-1)(N(x)-1+1)}{2} = \frac{N(x)(N(x)-1)}{2} = \binom{N(x)}{2}$$
$N(x)$ בחר 2 ללא חזרות - כפי שהיינו מצפים. כעת קל לראות כי $M = \max_{x \in G} P(x)$ וכן $P2 = \sum_{x \in G} P(x)$.
נחשב את הנתון לכל אחד מהסדרים:
1) nodeID:
   - $P2 = 3{,}975{,}462$
   - $M = 543{,}403$
2) (Degree, nodeID):
   - $P2 = 1{,}922{,}379$
   - $M = 7{,}750$
3) (5000-Degree,nodeID):
   - $P2 = 4{,}823{,}621$
   - $M = 545{,}490$

קל אם כן לראות כי בהתחשב בממצאים אליהם הגענו בשאלה 4, נרצה למזער את $P2$ ו-M ולכן נעדיף את הסידור ה-lexicography – (Degree, nodeID). באלגוריתם הסידורי נגיע אמנם לשיפור של פי 2~ בזמן הריצה, אך מבחינת המימוש השיפור בסיבוכיות זיכרון יכול להגיע עד לפי 70~. כמובן שבאלגוריתם ה-MapReduce השיפור רב יותר שכן קוראים וכותבים לא מעט פעמים וההשפעה של המקדם ניכרת מאוד!

התוצאה אמנם מאוד צפויה כיוון שגרף החברים מאוד דומה לגרף כוכב – מעט מאוד קודקודים שמקושרים להרבה – ועל כן מספור אשר נותן לכל אחד מהם ערך מאוד גבוה, מקטין מאוד את מספר המשולשים הפתוחים.

7. האלגוריתם פועל באופן הבא (אציין כי הוא מאוד מושפע מכך שבחרתי לממש אותו במטלב):
- לכל קודקוד נמצא את רשימת הקודקודים השכנים שלו הגדולים ממנו לפי רשימת הצלעות (פשוט כל הקודקודים בטור השני כאשר הוא בטור הראשון).
- לכל קודקוד i נעבור על רשימת הקודקודים השכנים הגדולים ממנו j:
  - נעבור על כל רשימת השכנים של i הגדולים מ-j, k:
    - אם k ברשימת השכנים של j (בפרט נבדקים רק הגדולים מ-j אך כמובן שמדובר ביתירות שכן k בודאות גדול מ-j על פי ההגדרה):
      - הוסף משולש לספירה

מספר המשולשים הוא אם כן 1,612,010 – נשים לב שהוא מהווה $\frac{1{,}612{,}010}{1{,}922{,}379} = 83.85\%$ מסך המשולשים הפתוחים כאשר הסדר המוגדר על הקודקודים הוא לקסיקוגרפי (Degree, nodeID). ניתן לומר אם כן כי זהו סידור מאוד אופטימלי.

8. בתחילה נראה כיצד ניתן ליצור את ה-sketches $S(i)$, בהינתן $h \sim U[0,1]$:

CreateSketch$(N(i), h, k)$:

1. $S(i) \leftarrow \emptyset$
2. לכל $j \in N(i)$:
   2.1. $jHash \leftarrow h(j)$
   2.2. אם $|S(i)| < k$:
      2.2.1. $S(i) \leftarrow S(i) \cup \{h(j)\}$
   2.3. אם $h(j) < \max(S(i))$:
      2.3.1. $S(i) \leftarrow S(i) \backslash \{max(S(i))\}$
      2.3.2. $S(i) \leftarrow S(i) \cup \{h(j)\}$
3. החזר את $S(i)$

כעת, נעבור לאלגוריתם אשר בהינתן ה-sketches ייתן לנו את ההערכה למספר המשולשים:

נעבוד לפי ההדרכה בתרגיל, בהינתן $(|N(i)|, |N(j)|, S(i), S(j), k)$:

EstimateIntersection$(|N(i)|, |N(j)|, S(i), S(j), k)$:

1. אם $|N(i)| \leq k$ וגם $|N(j)| \leq k$:
   1.1. החזר את $|S(i) \cap S(j)|$
2. אם $|S(i)| < k$:
   2.1. $\tau \leftarrow \max(S(j))$
3. אחרת אם $|S(j)| < k$:
   3.1. $\tau \leftarrow \max(S(i))$
4. אחרת:
   4.1. $\tau \leftarrow \min(max(S(i)), max(S(j)))$
5. $k' = |\{x \in S(i) \cup S(j): x \leq \tau\}|$
6. $jaccardEstimate \leftarrow \frac{|S(i) \cap S(j)|}{|S(i) \cup S(j)|}$
7. $unionEstimate \leftarrow \frac{k'-1}{\tau}$
8. החזר את $jaccardEstimate * unionEstimate$

נשים לב כי הערך של τ נכון כיוון שהחל משלב 2 רק אחד מהשניים יכול להתקיים

$S(i) < k \; or \; S(j) < k$ אך לא שניהם.

נבחין כי ה-$jaccardExtimator$ הוא מעצם הגדרתו מעריך של היחס $\frac{|N(i) \cap N(j)|}{|N(i) \cup N(j)|}$ ולכן כאשר נכפול אותו במעריך לגודל קבוצת האיחוד, $unionEstimator$, נקבל הערכה טובה לגודל קבוצת החיתוך.

9. כיוון שהמימוש הוא במטלב, והערכת $|N(i) \cap N(j)|$ לכל $i < j$ כך ש-$Friends(\{i, j\})$ דורשת מאיתנו לרוץ לכל קודקוד על כל השכנים הגדולים ממנו, זמן הריצה ארוך יותר, בין היתר עקב הכתיבה הרבה לזיכרון והשימוש בו על מנת לזכור את כלל ה-sketches, כלומר, בסך הכל החישוב כעת לוקח כ-42.65 במקום 29.49 שניות. כמובן, שזו לא אמורה הייתה להיות התוצאה ולמשל בפייתון התוצאה טובה יותר בכמעט פי שתיים.

ההערכה אליה הגענו במספר המשולשים 1,602,725, כלומר השגיאה הרלטיבית היא רק $relative\ error = 1 - \frac{1{,}602{,}725}{1{,}612{,}010} = 1 - 0.99424 = 0.00576$. במידה ואכן זמן הריצה קטן פי שתיים השימוש ב-sketches מאוד משתלם, יתר על כן מבחינת סיבוכיות בזכרון כיוון שלאחר חישובם סיבוכיות הזכרון במערכת היא רק $O(kn)$, השימוש בהם מאוד משתלם ככל ש-n גדל כמובן שאף ניתן למקבל את החישובים של ה-sketches ואז הרווח יהיה רב יותר.

10. אכן ניתן להעריך עבור כל קודקוד את כמות המשולשים אליהם הוא משתייך. נסתכל על שלשת קודקודים (x, y, z) כך שזהו משולש סגור. אזי עבור הקודקוד x, בסימון המתאים לשאלה 8, $y \in N(x) \cap N(z)$ וכן $z \in N(x) \cap N(y)$. כלומר, אילו נמצא במדויק את $|N(x) \cap N(z)|$ למעשה נספור כל משולש פעמיים. לכן עבור החישוב המדויק נקבל:

$$T(x) = \frac{1}{2} \sum_{z \in N(x)} |N(x) \cap N(z)|$$

את אלו, אנו כבר יודעים להעריך. למעשה, בסימון המתאים לשאלה 8, פשוט נריץ את האלגוריתם הבא: (נניח כי ה-sketches כבר קיימים בשלב זה):

TrianglesOfNode$\big(x, N(x)\big)$:

1. $T(x) \leftarrow 0$
2. לכל $y \in N(x)$:
   1. $T \leftarrow T + \text{EstimateIntersection}(|N(x)|, |N(y)|, S(x), S(y), k)$
3. החזר $\frac{T}{2}$.

נספח - הקוד

קוד שאלה 5

```
g = importdata('facebook_combined.txt');
b = histc(g,unique(g));
b = b(:,1)+b(:,2);
b = sort(b,'descend');
semilogy(b);
```

קוד שאלה 6

```
function []=Q6()
   g = importdata('facebook_combined.txt');
   entireAppearances = histc(g,unique(g));
   degrees = entireAppearances(:,1)+entireAppearances(:,2);

   onlyBiggerNeighbors = entireAppearances(:,1); % if x in column 1 he is smaller
   output(onlyBiggerNeighbors);

   degreesFirst(g, degrees);

   degrees = 5000-degrees;
   degreesFirst(g, degrees);
end

function []=degreesFirst(g, degrees)
   degreeNodeIds = zeros(length(unique(g)),1);
   for i = 1:length(degreeNodeIds)
      [x,y] = find(g==(i-1));
      iNeighbors = unique(g(x,:));
      for j = 1:length(iNeighbors)
         v = iNeighbors(j);
         if ((degrees(v+1)>degrees(i)) || (degrees(v+1)==degrees(i) && v>i-1))
            degreeNodeIds(i)=degreeNodeIds(i)+1;
         end
      end
   end
   output(degreeNodeIds);
end

function []=output(biggerNeighbors)
   PX = (biggerNeighbors.*(biggerNeighbors-1))/2;
   P2 = sum(PX)
   M = max(PX)
end
```

קוד שאלה 7

```
tic;
g = importdata('facebook_combined.txt');
counter = 0;
for i = 1:length(unique(g))
   biggerNeighbors = g(g(:,1)==i-1,2);
   for j = 1:length(biggerNeighbors)
      vj = biggerNeighbors(j);
      vjBiggerNeighbors = g(g(:,1)==vj,2);
      for k=j+1:length(biggerNeighbors)
         vk = biggerNeighbors(k);
         if(sum(vjBiggerNeighbors==vk))>0
            counter=counter+1;
         end
      end
   end
end
counter
toc
```

קוד שאלה 9

```
function T=Q9()
   tic;
   h = importdata('facebook_rand.txt');
   h = h(:,2);
   g = importdata('facebook_combined.txt');
   entireAppearances = histc(g,unique(g));
   degrees = entireAppearances(:,1)+entireAppearances(:,2);
   nodes = unique(g);
   k=10;
   sketches = -1*ones(length(nodes),k);
   for i = 1:length(nodes)
      [x,y] = find(g==(i-1));
      iNeighbors = unique(g(x,:));
      iNeighbors = iNeighbors(iNeighbors~=i-1);
      sketches(i,:) = CreateSketch(iNeighbors, h ,k);
   end
   T=0;
   for i = 1:length(nodes)
      biggerNeighbors = g(g(:,1)==i-1,2);
      for j=1:length(biggerNeighbors)
         vj = biggerNeighbors(j);
         T= T + EstimateIntersection(degrees(i),degrees(vj+1), sketches(i,:), sketches(vj+1,:),k);
      end
```

```
    end
    T=round(T/3);
    toc
end

function sketch=CreateSketch(neighbors,h,k)
    sketch = zeros(1,k);
    for i = 1:min(k,length(neighbors))
        vi = neighbors(i);
        sketch(i)=h(vi+1);
    end
    if(length(neighbors)<=k)
        return
    end
    for i = k+1:length(neighbors)
        vi = neighbors(i);
        hashVi = h(vi+1);
        if(max(sketch)>hashVi)
            sketch(sketch==max(sketch))=hashVi;
        end
    end
end

function intersectionSize = EstimateIntersection(iDegree, jDegree, iSketch, jSketch,k)
    iSketch = iSketch(iSketch~=-1);
    jSketch = jSketch(jSketch~=-1);
    if (iDegree <= k && jDegree <= k)
        intersectionSize = length(intersect(iSketch,jSketch));
        return
    end
    if iDegree<k
        tau = max(jSketch);
    elseif jDegree<k
        tau = max(iSketch);
    else
        tau = min(max(iSketch),max(jSketch));
    end
    unionSketches = union(iSketch, jSketch);
    kEffective = sum(unionSketches<=tau);
    sketchesIntersectionSize = length(intersect(iSketch,jSketch));
    unionSize = length(unionSketches);
    unionEstimate = (kEffective-1)/tau;
    intersectionSize = unionEstimate*sketchesIntersectionSize/unionSize;
end
```