digital whisper
Technologic Papers

פרדיגמת קל-כבד

מאת עמית ניר

הקדמה

במאמר קודם (תכנות דינמי, גיליון 117) דיברו על חשיבות יעילות זמן הריצה של אלגוריתמים בעולם האמיתי. בקצרה, רק כדי לתזכר למה יעילות של אלגוריתמים מעניינת אותנו, אשאל את השאלה הרטורית הבאה: "האם הייתם מוכנים לחכות דקה עבור כל חיפוש גוגל שהייתם עושים?" התשובה שלי היא ברור שלא. אם אתם כמוני, גם 10 שניות נשמע כמו נצח.

לכן, אני מקווה שביססנו, נרצה אלגוריתמים שרצים מהר ככל הניתן. במאמר זה אראה פרדיגמה (מויקיפדיה: "אקסמפלאר - אוסף דוגמאות משותפות של פתרונות של בעיות, המשמשות מופת ודגם לחיקוי כיצד יש לפתור בעיות דומות") הנקראית קל-כבד. שיטה זו מתייחסת לפיתרון בעיות בשני אופנים - באופן הקל, ובאופן הכבד. במאמר זה נראה בעיה, שלה שני אלגוריתמים שונים. האחד, מתאים למקרה קיצון ראשון, לו נקרא "הכבד" והשני, מתאים למקרה קיצון הפוך, לו נקרא "הקל". שילוב בין שני האלגוריתמים האלו עבור המקרים השונים יביא אותנו למימוש שיטת "קל-כבד" ולזמן ריצה טוב יותר מכל אחד מהאלגוריתמים בנפרד.

הגדרת הבעיה

הבעיה שלנו תעסוק בעולם הגרפים, אז ראשית, בואו נגדיר מהו גרף.

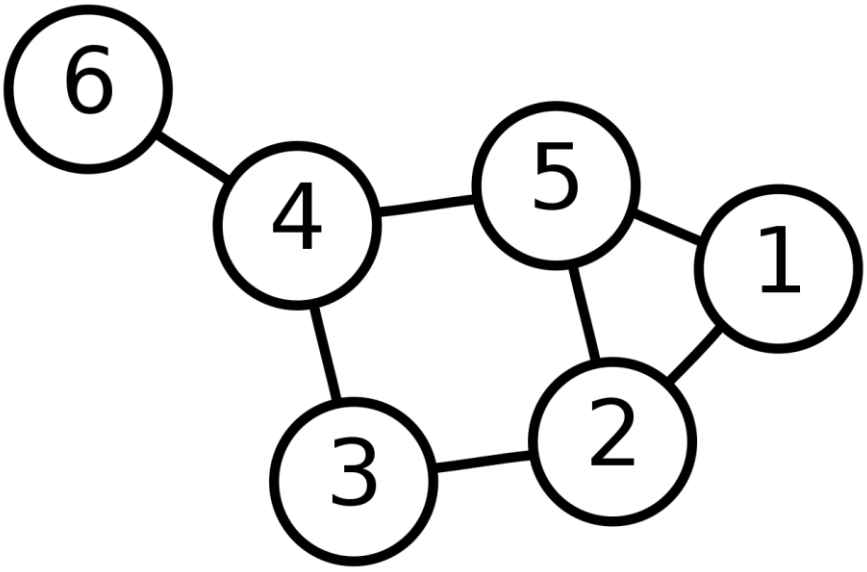
נסתכל על גרף כדוגמאות מהעולם האמיתי. גרף מורכב מקודקודים וקשתות, למשל - רשת הפייסבוק, נסתכל על פרופיל הפייסבוק שלכם. נתייחס לכל פרופיל פייסבוק כקודקוד, והקשתות יוגדרו לפי החברים שלכם - יש מכם קשת לכל אחד מחבריכם (קשת מסמלת "קשר חברות"). בגלל שבפייסבוק חברות היא קשר הדדי, כלומר, אם מישהו חבר שלכם אתם גם חברים שלו, נתייחס לקשתות כ"לא מכוונות", זאת אומרת, אין כיוון לקשר, זה לא שרק אחד מכם חבר של השני. דוגמה לגרף מכוון במציאות - רשת כבישים, יש לנו לעיתים כבישים חד סטריים שיהיו הקשתות המכוונות, והצמתים למשל ייצגו את הקודקודים. במאמר זה נתעסק בגרפים לא מכוונים בלבד.

יותר פורמלי, גרף (מסומל באות G הוא זוג קבוצות - קודקודים - (מסומל ב-V עבור Vertices) וקשתות (מסומל ב-E עבור Edges). כל קשת היא זוג של שני קודקודים מקבוצת הקודקודים V, למשל תסומן ב-(u,v) כאשר מתקיים ש-$u,v \in V$ (שני הקודקודים u,v שייכים (הסימון $\in$) ל-V). סך הכל, נסמן גרף כ-$G=(V,E)$. אנחנו נעסוק בגרפים לא מכוונים - כלומר, אין כיוון לקשתות. ניתן ללכת על כל קשת (u,v) גם בכיוון מ-u ל-v וגם בכיוון ההפוך (כמו שאתם חברים של כל חבר שלכם בפייסבוק).

נהוג לסמן קשתות מסוג זה בסוגריים מסולסלים {} מכיוון שאינם מעידים על סדר בעוד סוגריים עגולים () כן, אך מטעמי נוחות נסמן בסוגריים עגולים ונזכיר שוב שנעבוד רק על גרפים לא מכוונים.

להגדרה הפורמלית, מוזמנים להרחיב כאן.

ניתן דוגמה לגרף לא מכוון:



[הדוגמה לקוחה מויקיפדיה]

כאן, $G=(V,E)$ כאשר $V=\{1,2,3,4,5,6\}$ ו-$E=\{(4,6),(4,5),(4,3),(3,2),(2,5),(2,1),(1,5)\}$.

אנחנו נעסוק בבעיית דיווח על משולשים **בגרף לא מכוון** (אם תנסו לצייר משולש בגרף מכוון (כשיש כיוונים לקשתות) לא לגמרי ברורה ההגדרה, נכון? האם זה שכל הקשתות פונות לכיוון אחד שמשלים מעגל? האם אחת הקשתות יכולה להיות מנוגדת לכיוון של השתיים האחרות?), ולכן נבחר לעבוד על גרף לא מכוון.

משולש בגרף לא מכוון מוגדר בצורה הבאה: שלשת קודקודים $u,v,w \in V$ כך שכל הקשתות ביניהם קיימות - ובכתיב מתמטי $(u,v),(v,w),(w,u) \in E$, כלומר, כל הקשתות האפשריות ביניהן שייכות לקבוצת הקשתות. למשל, בדוגמה מלמעלה יש לנו משולש יחיד והוא מורכב מהקודקודים 1,2,5 מכיוון שכל הקשתות ביניהם קיימות. 1,2,3 הוא לא משולש למשל כי לא קיימת הקשת (1,3).

לסיכום נרצה לדווח על כל המשולשים בגרף לא מכוון (להחזיר רשימה של כל המשולשים הקיימים), כאשר לא יהיה לנו אכפת אם יהיו כפילויות ברשימה (יהיה ניתן להוריד אותן ללא פגיעה בזמן הריצה, לא נפרט על כך במאמר זה).

אלגוריתמים

כעת, נציג כמה אלגוריתמים שפותרים את הבעיה, נראה איזה הכי טוב ולאן נרצה להגיע.

אלגוריתם נאיבי

כנראה שאם תקדישו קצת מחשבה, תגיעו לאלגוריתם הנאיבי. לפי ויקיפדיה, ההגדרה לאלגוריתם נאיבי היא הבאה: "החיפוש הבסיסי ביותר הוא חיפוש בכוח גס - מעבר על כל הנתונים לפי סדרם, עד למציאת הנתון המבוקש, אם הוא אכן נמצא. מקובל לכנותו חיפוש ממצה בסיסי, או אלגוריתם חיפוש נאיבי. זהו חיפוש בלתי יעיל, וכאשר מספר הנתונים שבהם יש לחפש הוא גדול, החיפוש נמשך זמן רב, במידה בלתי סבירה". זאת אומרת, האלגוריתם הנאיבי ייעבור על כל האפשרויות למשולש, ויבדוק האם הוא אכן משולש - במידה וכן, נדווח עליו כמשולש.

הדרך הכי פשוטה לעבור על כל האפשרויות למשולש היא לעבור על כל שלשות הקודקודים $u,v,w \in V$ (השונים זה מזה), ועבור כל שלשה לראות אם שלושת הקשתות קיימות. במאמר זה לא אסביר איך עושים את הבדיקה הזו מכיוון שצריך להיכנס לדרכי ייצוג גרפים. לשם הפשטות נניח שהיא מתבצעת בזמן קבוע, כלומר $O(1)$ פעולות. להרחבה על הסימונים בהם נשתמש לסימול זמן ריצה O,Ω,Θ, קראו פה.

Algorithm 1 Print all triangles in a Graph $G=(V,E)$

```
output ← {}
for every u, v, w ∈ V do
    if u ≠ v ≠ w and (u, v) ∈ E and (w, u) ∈ E and (w, v) ∈ E then
        output ← output + (u, v, w)
```

ניתן לראות את מימוש האלגוריתם בפסאודו קוד (קוד גנרי שלא כתוב באף ספסיפיקציה של שפה מסוימת).

כעת, נרצה להבין מה זמן הריצה של האלגוריתם. בשביל המשימה הזו, נראה שהתנאי וההדפסה נעשים במספר קבוע של פעולות, ולכן יעניין אותנו רק כמה פעמים הלולאה נקראת. נבחין כי הלולאה נקראת כמספר שלשות הקודקודים השונים, שזה בדיוק אומר כמות האופציות לבחור 3 איברים מקבוצה בגודל $|V|$ (ערך מוחלט מסמל גודל של קבוצה). בכתיב מתמטי, נכתוב זאת כ-$\binom{|V|}{3}$ (יקרא כ-$|V|$ צ'וז 3), לעוד הרחבה על הנושא, מוזמנים לקרוא כאן.

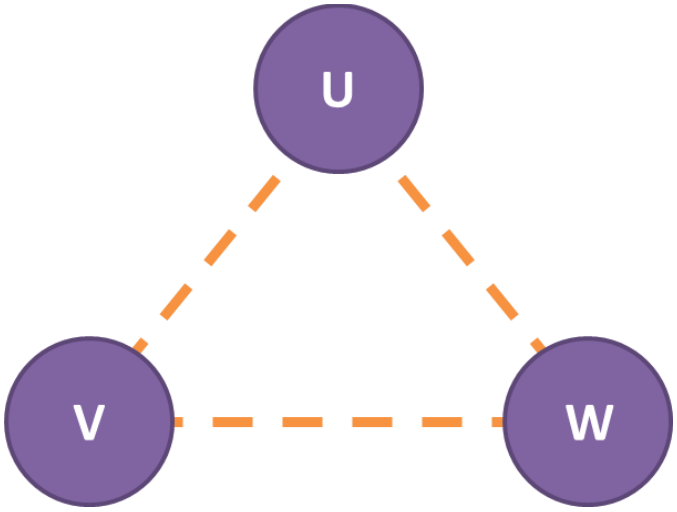
לפי נוסחת הצ'וז, אנו יודעים שזמן הריצה שלנו יוצא $O(|V|^3)$, שכפי שאנחנו מבינים זה בדיוק לעבור על כל שלשות הקודקודים האפשריות, כאשר כמו בבסיסי ספירה, יש לנו "3 ספרות" שלכל אחת יש $|V|$ אפשרויות, ולכן זה הגיוני.

אלגוריתם שני

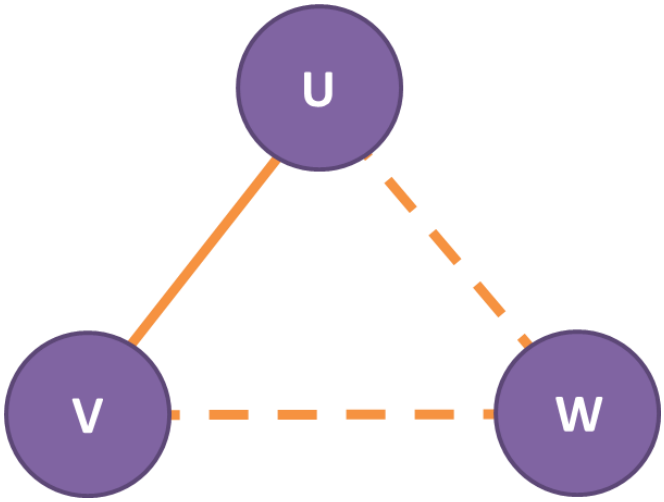
נשים לב, שזמן הריצה שלנו תלוי בקבוצת הקודקודים בלבד. נראה אלגוריתם שעושה שימוש גם בקבוצת הקשתות כעת כחלק מזמן הריצה.

במקום לקחת שלשת קודקודים ולבדוק האם קיימות שלושת קשתות בין כל שלושת הקודקודים, ניקח כל זוג של קודקוד $w \in V$ וקשת $(u,v) \in E$ אפשריים ונבדוק האם שתי הקשתות בין קצוות הקשת לקודקוד הנבחר קיימות.

תרשים למקרה הקודם:



תרשים למקרה הנוכחי:



אנחנו נבדוק האם הקשתות המקווקות קיימות. באלגוריתם הראשון - נצטרך לבדוק את כל שלושת הקשתות כי נתונים לנו שלושת הקודקודים. באלגוריתם השני, נצטרך לבדוק את שתי הקשתות הנותרות המורכבות מהקודקוד הנתון וקצוות הקשת הנתונה.

Algorithm 2 Print all triangles in a Graph $G = (V, E)$

```
output ← {}
for every w ∈ V do
    for every (u, v) ∈ E do
        if u ≠ v ≠ w and (w, u) ∈ E and (w, v) ∈ E then
            output ← output + (u, v, w)
```

כעת, ננתח זמן ריצה. שוב, נבחין כי התנאי וההדפסה בו לוקחות מספר קבוע של פעולות ומה שישפיע לנו על זמן ריצה זה כמה פעמים אנו בודקים את התנאי, כלומר - כמה פעמים אנו מגיעים לתוך הלולאה הפנימית. מספר זה שווה לכמות הפעמים שהלולאה החיצונית מתבצעת כפול כמות הפעמים שהפנימית מתבצעת מכיוון שעבור כל ריצה של החיצונית, הפנימית מתבצעת במלואה, וסך הכל זה לסכום את הפנימית, חיצונית פעמים. למי שזה היה יותר מדי - נחזור לפשוט והמוכר - פנימית כפול חיצונית, כלומר $O(|V| \cdot |E|)$ - החיצונית מתבצעת כמספר הקודקודים $|V|$ והפנימית עוברת על כל הקשתות, ולכן כמספר הקשתות, $|E|$.

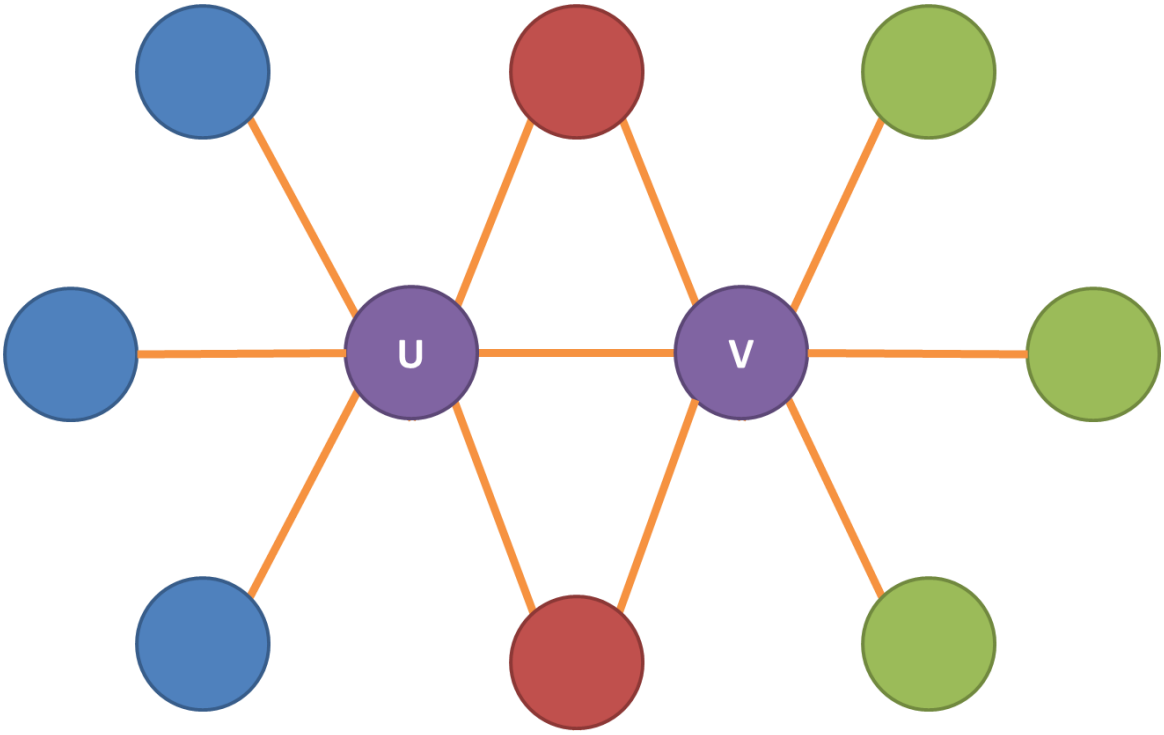
איך זמן הריצה הזה ביחס לזמן הריצה ממקודם? נבחין כי זמן הריצה הזה אף פעם לא יהיה יותר גרוע מזמן הריצה של האלגוריתם הראשון, מכיוון שבגרף "הכי גרוע" - כלומר, הכי צפוף, עם הכי הרבה קשתות, $|E| = O(|V|^2)$ (תחשבו על המקרה בו כל הקשתות קיימות, יש $\binom{|V|}{2} = O(|V|^2)$ קשתות בגלל שלכל קודקוד יכולות להיות לכל היותר $|V| - 1$ קשתות, אחת לכל קודקוד אחר, ויש לנו $|V|$ קודקודים), זמן הריצה זהה. אך בהמון מקרים אחרים בהם יש פחות קשתות, זמן הריצה יהיה נמוך יותר. לכן נעדיף את אלגוריתם 2, מכיוון שהוא אף פעם לא יהיה גרוע יותר מ-1.

אלגוריתם שלישי

כעת, נשים לב שבאלגוריתם הקודם יכלנו להפוך את הסדר בין הלולאות, ולהתייחס לכך במקום "לכל קודקוד לעבור על כל הקשתות" כ-"לכל קשת לעבור על כל הקודקודים", נסכים שזה שקול ולא משנה על נכונות וזמן הריצה של האלגוריתם, זו רק החלפה בין שורות הלולאות. כעת, נשאלת השאלה הבאה - האם באמת לכל קשת צריך לעבור על כל הקודקודים בגרף? התשובה היא לא. מספיק לעבור על השכנים של שני קצוות הקשת בלבד (שכנים של קודקוד מסוים הם כל מי שיש קשת ביניהם לבין אותו הקודקוד). למה זה מספיק? כי נבחין שאם קודקוד מסוים בגרף, למשל x לא שכן של אחד מקצוות הקשת, למשל u, אז בהכרח לא קיימת הקשת ביניהם ובהכרח לא יוכלו להרכיב משולש!

יותר מכך, נשים לב שאפילו אין צורך לבדוק האם קשתות קיימות באלגוריתם מהסוג הזה - עבור כל קשת (u,v), במקום לעבור על כל הקודקודים w כמו מקודם ולבדוק, נוכל פשוט לחשב את המשותף בין קבוצות השכנים של u ושל v! נסמן את קבוצת השכנים של קודקוד u ב-$N(u)$. למה זה נכון? כי אם קודקוד w נמצא גם ב-$N(u)$ וגם ב-$N(v)$, הרי שהקשתות (u,w) ו-(v,w) קיימות, כי הוא שכן של שניהם, וברור שהקשת (u,v) קיימת מכיוון שממנה התחלנו.

נמחיש בציור:



הקודקודים המסומלים בכחול הם השכנים של u, בירוק השכנים של v, ובאדום השכנים של שניהם - אלו היחידים שמרכיבים לנו משולש.

ולכן, נסכם לכדי האלגוריתם הבא:

Algorithm 3 Print all triangles in a Graph $G = (V, E)$

```
output ← {}
for every (u, v) ∈ E do
    for every w ∈ N(u) and w ∈ N(v) do
        output ← output + (u, v, w)
```

לפני שננתח זמן ריצה נגדיר סימון מתמטי חדש - האות היוונית הגדולה סיגמה שמסומלת באות Σ, שמסמלת סכום, למשל: $\sum_{i=1}^{3} i = 6$, שזה אומר: תסכום את הביטוי i, כאשר i הולך מ1 עד 3. אפשר להתייחס לזה בתור לולאת פור שעוברת על ה-range של הערכים שכתובים מהערך התחתון שמתחת לסיגמה עד לערך העליון שמעל הסיגמה, ומבצעת סכימה על הערך שרשום במרכז (באותה מידה זה היה יכול להיות ביטוי יותר מסובך כמו i^2, ואז התוצאה הייתה יוצאת $1^2 + 2^2 + 3^2 = 14$).

כעת, זמן הריצה שלנו יהיה עבור כל קשת, חישוב המשותף בין רשימות השכנים של קצוותיה. בהנחה שהרשימות ממוינות (שוב, יצוג של גרפים), ניתן לחשב באופן דיי מגניב את החיתוך בין הרשימות בזמן ריצה של חיבור אורכי הרשימות, לא אפרט כאן איך, אך יותר ממוזמנים לחשוב על כך, תרגיל ממש נחמד, ולמי שרוצה את הפיתרון - ניתן למצוא אותו כאן.

ולכן, זמן הריצה הכולל שלנו יהיה:

$$O\left(\sum_{(u,v)\in E} \deg(u) + \deg(v)\right)$$

רגע רגע רגע, מה כתוב כאן?

אוקיי אז מה שכתוב בסיגמה זה בעצם לעבור על כל הקשתות האפשריות - ולסכום את הביטוי שיש בפנים, נוטציה קצת שונה מזאת שראינו מקודם, אך מכיוון שיותר פשוט לכתוב זאת ככה במקום להתייחס לקבוצת הקשתות כרשימה ולעבור מ-$i = 0$ עד אורך הרשימה וכל פעם לכתוב בפנים שאנחנו מתייחסים לאיבר הi, שהוא (u, v), ... מסורבל יותר, לא? בקיצור, זה אומר לסכום את הביטוי שיש בפנים על כל הקשתות הקיימות, למי שלא עקב.

מה שרשום בתוך הביטוי זה מה שדיברנו עליו מקודם - גודל רשימת השכנים של u, זו בעצם הדרגה של u, שמסומנת באותיות deg עבור degree. ולכן, אנחנו סוכמים לכל קשת את הדרגה של של קודקודיה - שזה שווה ללסכום עבור כל קשת את גודל רשימות השכנים של שני קודקודיה.

הבחנת יתרונות

נשווה בין אלגוריתם 3 לאלגוריתם 2 - נבחין כי כמו שאלגוריתם 2 לא יותר גרוע מאלגוריתם 1, כך אלגוריתם 3 לא יותר גרוע מאלגוריתם 2. מכיוון שלכל קשת אנחנו עובדים כמות זמן ששווה לסכום דרגות הקודקודים שלה, שבמקרה הגרוע יהיה $O(|V|)$ (למשל אם אחד הקודקודים מחובר לכל שאר הקודקודים בגרף), אך בהרבה מקרים אחרים יהיה פחות (כל אחד מהקודקודים מחובר לעוד קודקוד אחד בלבד למשל). לעומת זאת, באלגוריתם 2, עבור כל קשת תמיד נבצע $O(|V|)$ פעולות, מכיוון שתמיד עבור כל קשת אנחנו עוברים על כל הקודקודים.

אז... למה עשינו את כל התהליך הזה? למה לא ישר דיברנו על אלגוריתם 3 אם נעדיף אותו תמיד..?

אז נבחין שלפעמים עוזר להסתכל על הדברים מזווית קצת שונה, ואם הסתכלנו על הדברים מהזווית של הקשתות, וכמה פעולות אנחנו מבצעים עבור כל קשת, נסתכל על הדברים מזווית הקודקודים, וכמה פעולות אנחנו מבצעים עבור כל קודקוד.

אז באלגוריתם 2 מאוד פשוט לנתח את זה, עבור כל קודקוד אנחנו מבצעים $O(|E|)$ פעולות, ניתן לראות את זה לפי הלולאות ממקודם, עבור כל קודקוד אנחנו עוברים על כל הקשתות.

באלגוריתם 3, טיפה יותר מסובך לנתח את זה, אבל אנחנו נצליח, לא לדאוג!

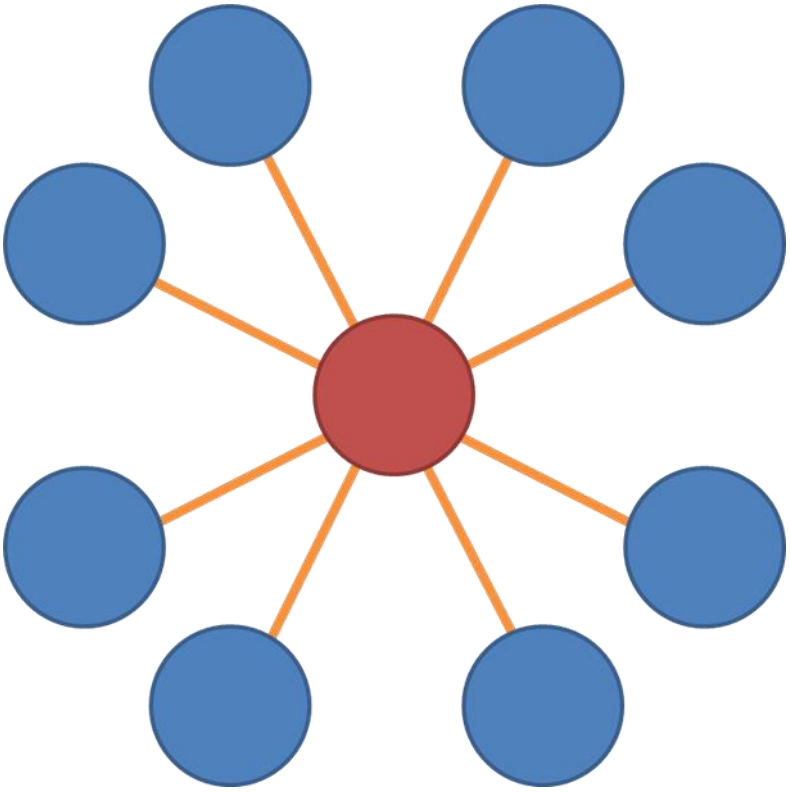
נשים לב, שעבור כל פעם שקודקוד u משתתף בקשת, הוא תורם $O(\deg(u))$ לזמן הריצה. נשאלת השאלה, בכמה קשתות קודקוד משתתף? התשובה היא, $\deg(u)$ קשתות. למה? כי הדרגה מסמלת את מספר השכנים שלו - ומספר השכנים שלו שווה למספר הקשתות בו הוא משתתף כי כל שכן מעיד על קשת. לכן סך הכל באלגוריתם 3 כל קודקוד תורם $O(\deg^2(u))$ לזמן הריצה.

בואו נסתכל על שני מצבי קיצון - קודקוד שמקושר להמון קודקודים (דרגה גבוהה), וקודקוד שמקושר לממש מעט (דרגה נמוכה). נבחין שעבור קודקודים מדרגה נמוכה, נעדיף לשלם עליהם את דרגתם בריבוע (להשתמש באלגוריתם 3), ולא את מספר הקשתות, מכיוון שדרגתם קטנה. עבור קודקודים מדרגה גבוהה נעדיף לשלם עליהם $O(|E|)$ פעם אחת (להשתמש באלגוריתם 2), ולא את דרגתם בריבוע מכיוון שזה יכול לעלות לנו המון.

איך נעשה את זה? על קודקודים מסוגים שונים נפעיל את שני האלגוריתמים השונים! נפרט על כך בהמשך, כעת, נדבר על זמן הריצה שאנחנו רוצים להגיע אליו.

אך לפני כן, נמחיש את מה שדיברנו עליו.

למשל, בתרשים הנ"ל, על הקודקוד האמצעי (אדום) נעדיף לשלם פעם אחת $O(|E|)$ וזהו, כי לשלם את דרגתו בריבוע יעלה לנו $O(|E|^2)$, בגלל שהוא מחובר לכל שאר הקודקודים ואין עוד קשתות בגרף ולכן דרגתו שווה ל-$|E|$. לעומת זאת, על כל אחד מהקודקודים "הקיצוניים" (כחולים) נעדיף לשלם את דרגתם בריבוע מכיוון שדרגתם שווה ל-1 (משתתפים רק בקשת אחת), במקום לשלם עליהם $O(|E|)$.



כמובן שזו רק דוגמה להמחשה על ההעדפה שלנו, ושבגרף הזה אין משולשים. אם היינו מחברים כל שני קודקודים "קיצוניים" צמודים ליצירת מעין מעגל עם מרכז, אז היו לנו הרבה משולשים, שכל אחד מורכב משני קודקודים קיצוניים ומהקודקוד המרכזי, אבל עדיין דרגת כל קודקוד קיצוני הייתה $3 = O(1)$ לעומת הקודקוד האמצעי שדרגתו עדיין הייתה שווה ל$O(|E|)$. כמובן שמדובר בדוגמה להמחשה וניתן להרחיב אותה ולבנות גרף דומה עם 10000 קודקודים במעגל החיצוני במקום 8.

זמן אידיאלי

ראינו, שככל שאנחנו מתעסקים יותר ויותר בפרמטר הקשתות, אנחנו מגיעים לזמני ריצה יותר טובים. אמנם במקרה הגרוע הם זהים לנאיבי - אבל בהמון מקרים אחרים עבור גרפים "דלילים" יותר, אלגוריתמים שתלויים בפרמטר הקשתות פועלים בזמנים יותר טובים.

לכן - באידיאל נרצה להגיע לזמן ריצה שתלוי רק בפרמטר הקשתות, איזה זמן ריצה זה יהיה?

בשביל לנתח את זמן הריצה שלנו - אנחנו צריכים להסתכל על המקרה הגרוע ביותר. אז, נסתכל על הגרף הגרוע ביותר - בו כל הקשתות קיימות - $|E| = O(|V|^2)$. במקרה הזה, כל המשולשים קיימים, ולכן זמן הריצה שלנו יהיה לפחות $O(|V|^3)$, מכיוון שרק להדפיס את כל המשולשים ולהחזיר את הפלט יקח כמספר המשולשים. לכן, אם נרצה לבטא את זמן הריצה שלנו כתלות בפרמטר הקשתות בלבד, זמן הריצה שלנו יהיה $O(|E|^{1.5}) = O(|V|^3)$. זה זמן הריצה שנשאף אליו. נכון שבמקרה המדובר זמן הריצה משתווה לזמן הריצה של האלגוריתם הנאיבי, אבל כמו באלגוריתמים 2 ו-3 שראינו, בהמון מקרים אחרים ובהמון גרפים אחרים יותר "דלילים", יהיה הרבה יותר יעיל.

לסיכום, נרצה להגיע לזמן ריצה של $O(|E|^{1.5})$, באידיאל.

שילוב בין האלגוריתמים עם שיטת קל כבד

אז אמרנו, עבור קודקודים עם דרגה נמוכה - "קלים", נרצה לעבור עבורם דרגתם בריבוע - כלומר, להפעיל את אלגוריתם 3. אבל עבור קודקודים עם דרגה גבוהה - "כבדים", לא נרצה לשלם את דרגתם בריבוע, אלא $|E|$ בלבד, ולכן נרצה להפעיל עליהם את אלגוריתם 2.

בטח יש כמה שאלות שלא ברורות לכם עדיין והן:

איך אנחנו מגדירים דרגה נמוכה ודרגה גבוהה?

שאלה מצויינת, שמחה ששאלתם! נגדיר קודקוד קל כקודקוד שדרגתו קטנה שווה לשורש מספר הקשתות, כלומר קודקוד u יקרא קל אם $\deg(u) \leq \sqrt{|E|}$. נגדיר קודקוד כבד כקודקוד שדרגתו גדולה משורש מספר הקשתות, כלומר, קודקוד u יקרא כבד אם $\deg(u) > \sqrt{|E|}$. בהמשך אסביר למה בחרתי דווקא בשורש מספר הקשתות.

עוד שאלה שבטח שאלתם את עצמכם היא מה זאת אומרת להפעיל את האלגוריתמים על קודקודים? הרי הם מקבלים גרף ומחזירים משולשים...

ואתם צודקים לחלוטין. בשביל זה, נגדיר עוד שתי הגדרות נוספות, משולש קל ומשולש כבד.

משולש קל הוא משולש שכל קודקודיו קלים, ומשולש כבד הוא משולש שלפחות אחד מקודקודיו כבדים. נרצה למצוא את כל המשולשים הכבדים עם אלגוריתם 2, ואת כל המשולשים הקלים עם אלגוריתם 3.

איך נעשה את זה? באלגוריתם 2, לפני שנעבור על כל הקשתות, נוסיף את הבדיקה אם הקודקוד שלנו הוא כבד! כך - נמצא את כל המשולשים הכבדים, כי נעבור על כל הקודקודים הכבדים, ועבור כל קודקוד כבד נעבור על כל הקשתות.

באלגוריתם 3, נרצה לעבור על כל המשולשים הקלים, נעשה את זה על ידי שנוסיף בדיקה, לפני שנחשב את המשותף בין שתי רשימות השכנים של הקודקודים, נבדוק ששניהם קלים, וכך, נמצא את כל המשולשים הקלים (ניתן להוסיף את הבדיקה גם עבור הקודקודים ברשימה המשותפת לאחר מכן, אך זה מיותר (מוזמנים לחשוב למה!), בקצרה, יחזרו לנו גם ככה כל המשולשים הקלים, וגם קצת כבדים, והרי אמרנו שלא מעניין אותנו כפילויות).

כך, סך הכל, נחזיר את כל המשולשים! עכשיו, אולי גם יותר מובן למה הגדרנו משולש קל כמשולש שכל קודקודיו קלים ולא משולש שאחד מקודקודיו קל, כי אם היינו מגדירים כך, אז באלגוריתם 3 היינו עובדים מאוד קשה לחישוב החיתוך (הרשימה המשותפת), בעוד שלפי ההגדרה שלנו, אנחנו עובדים לכל היותר $2\sqrt{|E|}$, מכיוון ששני הקודקודים קלים אזי הדרגה של כל אחד מהם היא לכל היותר $\sqrt{|E|}$.

נכתוב את האלגוריתם בפסאודו קוד:

Algorithm 4 Print all triangles in a Graph $G = (V, E)$

```
output ← {}
for every w ∈ V do
    determine if w is heavy or light
make two groups accordingly: V_heavy, V_light
for every w ∈ V_heavy do
    for every (u, v) ∈ E do
        if u ≠ v ≠ w and (w, u) ∈ E and (w, v) ∈ E then
            output ← output + (u, v, w)
for every (u, v) ∈ E where u ∈ V_light and v ∈ V_light do
    for every w ∈ N(u) and w ∈ N(v) do
        output ← output + (u, v, w)        ▷ here we can add "if w ∈ V_light"
```

כעת, מה זמן הריצה?

הלולאה הראשונה יכולה להתבצע ב-$O(|V| + |E|)$, על ידי סריקה פשוטה של הגרף - לכל קודקוד לעבור על כל הקשתות שלו.

הלולאה השנייה, מתבצעת ב-$O(|V_{heavy}| \cdot |E|)$. השאלה היא, מהו $|V_{heavy}|$? אסביר למה הוא לכל היותר $2 \cdot \sqrt{|E|}$. נניח שהוא היה יותר, כלומר, לפחות $2 \cdot \sqrt{|E|} + 1$. סכום הדרגות של הקודקודים הכבדים בלבד הוא לפחות $\sqrt{|E|} \cdot \left(2 \cdot \sqrt{|E|} + 1\right) = 2 \cdot |E| + \sqrt{|E|} > 2 \cdot |E|$ (הדרגה המינימלית של קודקוד כבד כפול מספר הקודקודים הכבדים). אבל נשים לב, שסכום דרגות כל הקודקודים בגרף (קלים וכבדים יחד) הוא רק $2 \cdot |E|$, בגלל שכל קשת נספרת פעמיים (נספרת פעם אחת עבור כל אחד מהקודקודים שבקצוות שלה, ולכן פעמיים). כלומר, המצב שיצרנו לא אפשרי! אז בהכרח יש לכל היותר $2 \cdot \sqrt{|E|}$ קודקודים כבדים לכל היותר! הוכחה זו אגב, היא מסוג "הנחה בשלילה", בה אנחנו רוצים להוכיח

משהו אחד, אז מניחים שהוא לא נכון ובעזרת מעברים חוקיים מגיעים לסתירה - למשהו שלא היה יכול להתקיים, ולכן היינו חייבים להתחיל ממשהו לא נכון בהתחלה כדי להגיע למצב הזה, והוכחנו את מה שרצינו. למי שלא עקב, הוכחנו ש-$|V_{heavy}|$ הוא לכל היותר $2\sqrt{|E|}$, ולכן הלולאה השנייה מתבצעת סך הכל ב-$O\left(2\sqrt{|E|}\cdot|E|\right)=O(|E|^{1.5})$.

הלולאה השלישית עוברת על כל קשת, ואם שני קודקודיה קלים מבצעת כסכום הדרגות שלהם. כעיקרון, יכול להיות גרף שכל הקודקודים בו קלים, ולכן זמן הריצה של הלולאה הזו במקרה הגרוע הוא גם $O(|E|^{1.5})$, מכיוון שיש לנו $|E|$ קשתות ובמקרה הגרוע לכל קשת מבצעים $O(\sqrt{|E|})$ פעולות (לחשב את החיתוך בין הרשימות עולה כסכום הדרגות שהוא $O(\sqrt{|E|})$ כי שני הקודקודים קלים).

סך הכל הגענו לזמן ריצה מאוד דומה למה שרצינו להגיע אליו מקודם! $O(|V|+|E|^{1.5})$! ניתן לחשוב כתרגיל מחשבה למה לא יכלנו לוותר על ה-$|V|$ הזה, ולמה זה לא כואב לנו שהוא נמצא שם.

סיכום

אלגוריתמים זה מגניב!

ראינו כיצד לקחנו שני אלגוריתמים יחסית פשוטים, ובעזרת הסתכלות עליהם מזווית קצת שונה והבחנה ביתרונות ובחסרונות של כל אחד הצלחנו לשלב ביניהם לכדי אולטרה-אלגוריתם שהוא יותר טוב מכל אחד מהם בנפרד!

חובה שלי אליכם - למה בחרנו דווקא ב-$\sqrt{|E|}$? איך ידעתי שזה מה שיצא לנו בול מה שצריך? אז התשובה היא, שלא ידעתי. כעיקרון, הייתי צריכה להגדיר את ההגדרה לקודקודים קלים וכבדים עם פרמטר גנרי לא ידוע, ולנתח את כל זמן הריצה באמצעותו, ואז למצוא למה הוא צריך להיות שווה כדי שיהיה לנו הכי טוב ומשתלם - ואכן יצא $\sqrt{|E|}$. מוזמנים לנסות בעצמכם כמובן.

המאמר מבוסס על רעיונות שהציגו פרופסור אלון איתי ופרופסור מיכאל רודה בשנת 1978 במאמרם:
http://www.cs.technion.ac.il/~itai/publications/Algorithms/min-circuit.pdf

ארצה להודות לחברי שעזרו לי ועברו על המאמר טרם פרסומו, ובמיוחד ל**דור סלומון**, **יהונתן ליפשיץ**, **רואי פלדמן**, **איתי גרדנויץ**, **ניתאי רייטר**, **איתי הוך** ו**רון שרמן**.

קצת על עצמי

לפני גיוס בעת כתיבת שורות אלה, כנראה שכשיתפרסמו אהיה על מדים. נכון לעכשיו מתרגלת את הקורס "אלגוריתמים 1" באוניברסיטת בר אילן, ועובדת בחברת "סייבריליום" כ-Cyber Security Expert עד הגיוס. מאוד אוהבת את התחום של האלגוריתמיקה, ומתכננת יום אחד לעשות תואר שני בנושא.