

**האוניברסיטה הפתוחה**

**המחלקה למתמטיקה ולמדעי המחשב**

**החטיבה למדעי המחשב**

פרוייקט מתקדם במדעי המחשב

מימוש פתרון להתאמת תבניות ב- Neo4j והשוואתו לשימוש באינדקס gIndex

מנחה: פרופ' אהוד גודס

מגיש:

מיכאל וייסברג

ת.ז. 314342296

סמסטר ב' 2017

תוכן עניינים

[גרסאות המסמך 4](#_Toc497582433)

[1. תקציר 5](#_Toc497582434)

[2. מטרת הפרוייקט 5](#_Toc497582435)

[3. מבוא 6](#_Toc497582436)

[3.1. בסיסי נתונים גרפיים 7](#_Toc497582437)

[3.1.1. מהו בסיס נתונים גרפי? 7](#_Toc497582438)

[3.2. הגדרה פורמלית של הבעיה 7](#_Toc497582439)

[3.2.1. עיבוד שאילתא גרפית (pattern matching) 7](#_Toc497582440)

[3.2.2. האלגוריתם הנאיבי 7](#_Toc497582441)

[3.2.3. גישת אינדוקס המסלולים\* 8](#_Toc497582442)

[3.2.4. גישת אינדוקס תת-גרפים 9](#_Toc497582443)

[3.2.5. גישת אינדוקס תת-עצים 10](#_Toc497582444)

[3.3. מינוחים והגדרות 10](#_Toc497582445)

[3.3.1. גרף לא מכוון עם תוויות 10](#_Toc497582446)

[3.3.2. תת-גרף 10](#_Toc497582447)

[3.3.3. גרף-על 10](#_Toc497582448)

[3.3.4. איזומורפיזם תת-גרפים 10](#_Toc497582449)

[3.3.5. ייצוג קנוני/תווית קנונית של גרף 10](#_Toc497582450)

[3.4. תיאור האלגוריתם עצמו - gIndex 12](#_Toc497582452)

[3.4.1. מציאת קבוצת פריטים discriminative שכיחים 12](#_Toc497582454)

[3.4.2. העץ gIndex tree 13](#_Toc497582455)

[3.4.3. שלב החיפוש 13](#_Toc497582456)

[3.4.4. שלב האימות 14](#_Toc497582457)

[3.5. מהו Neo4j 14](#_Toc497582464)

[3.5.1. מאפייני ה-ACID 14](#_Toc497582465)

[3.5.2. שפה ייעודית 14](#_Toc497582466)

[3.6. מבנה העבודה 17](#_Toc497582467)

[3.6.1. בעיית התאמת התבניות ב- Neo4j 17](#_Toc497582468)

[3.6.2. מימוש gIndex [4] 17](#_Toc497582469)

[3.6.3. בסיס נתונים לבדיקה 17](#_Toc497582470)

[3.7. סביבת העבודה 17](#_Toc497582471)

[3.8. התזה 17](#_Toc497582472)

[4. מימוש הפרוייקט 18](#_Toc497582473)

[4.1. תיאור ומבנה כללי 18](#_Toc497582474)

[4.2. פרוייקט Common 19](#_Toc497582476)

[4.2.1. מחלקה DIFactory 19](#_Toc497582477)

[4.2.2. מחלקות ה-Logger-ים 20](#_Toc497582478)

[4.2.3. מחלקה EnumerableComparer 20](#_Toc497582479)

[4.3. הפרוייקט Model 20](#_Toc497582480)

[4.3.1. מחלקה Graph 21](#_Toc497582481)

[4.3.2. מחלקה Node 21](#_Toc497582482)

[4.3.3. מחלקה DFS\_Code 21](#_Toc497582483)

[4.3.4. מחלקה Neighbor 21](#_Toc497582484)

[4.3.5. מחלקה Isomorphism 21](#_Toc497582485)

[4.4. פרוייקט DAL 21](#_Toc497582486)

[4.4.1. מחלקה Neo4jDAL 22](#_Toc497582487)

[4.4.2. מחלקות SyntheticGraphDatabaseReader ו- SyntheticGraphDatabaseWriter 23](#_Toc497582488)

[4.4.3. מחלקה FrequentFeaturesFileDal 23](#_Toc497582489)

[4.4.4. מחלקה GraphDatabaseCsvWriter 23](#_Toc497582490)

[4.4.5. מחלקה EdgePathToCypherQueryConverter 23](#_Toc497582491)

[4.5. פרוייקט BL 23](#_Toc497582492)

[4.5.1. מחלקה FrequentFeatureSelector 24](#_Toc497582493)

[4.5.2. מחלקה PathFinder 24](#_Toc497582494)

[4.5.3. מחלקה GraphPathsGenerator 24](#_Toc497582495)

[4.5.4. מחלקה SubgraphIsomorphismGenerator 24](#_Toc497582496)

[4.5.5. מחלקה SyntheticGraphDatabaseLoader 25](#_Toc497582497)

[4.5.6. מחלקה PatternMatcher 25](#_Toc497582498)

[4.6. פרוייקט Neo4jSyntheticDatabaseFiller 25](#_Toc497582499)

[4.7. פרוייקט FrequentFeaturesFileWriter 25](#_Toc497582500)

[4.8. פרוייקט GIndexVsNeo4jRunner 26](#_Toc497582501)

[5. תוצאות 27](#_Toc497582502)

[6. מסקנות 28](#_Toc497582503)

[7. ביבליוגרפיה 29](#_Toc497582504)

# גרסאות המסמך

|  |  |  |
| --- | --- | --- |
| **מס' גירסה** | **שינויים** | **תאריך** |
| **1.0** | טיוטה ראשונה | 11.10.2017 |
|  |  |  |
|  |  |  |
|  |  |  |

# תקציר

הגדרה: התאמת תבניות

בהינתן מסד נתונים גרפי D, ושאילתא q שהיא גרף בעצמה, מצא את כל הגרפים כך, שהשאילתא q היא תת-גרף בכל אחד מהם. השאילתא q יכולה להיות כל משפט sql-י, בהתייחס לגרפים.

בהינתן בסיס נתונים גרפי, נרצה לבצע עליו שאילתות שונות ומגוונות, בכדי למצות את המידע המוכל בו. בדומה לבסיס נתונים רלציוני קלאסי, הרצת שאילתות נאיבית עלולה להיות לא יעילה, ולכן נרצה לייעל את עיבוד השאילתא הגרפית במידת האפשר. בבסיסי נתונים רלציוניים קלאסיים, ייעול שאילתות נעשה ע"י בניית אינדקסים מתאימים על המידע בבסיס הנתונים והרצת השאילתא עליו. נרצה לעשות תהליך דומה גם כאן.

בפרוייקט זה מימשתי פתרון להתאמת תבניות מעל בסיס נתונים מסוג Neo4j ע"י פירוק שאילתות למסלולים וע"י מימוש אינדקס מסוג gIndex [4] מעל בסיס הנתונים הנ"ל, וזאת בכדי להוכיח שמימוש אינדקס מעל בסיס הנתונים ייתן תוצאות טובות יותר מפתרון הפירוק למסלולים בעבור שאילתות גדולות.

הדו"ח הנ"ל יתאר את-

1. מטרת הפרוייקט
2. עולם הבעיה, המונחים והגישות לפתרון
3. האלגוריתם gIndex כמו שהוא מוצג במאמר [4]
4. המימוש שלי לאלגוריתם gIndex ולהתאמת תבניות בשתי הדרכות שצויינו לעיל מעל Neo4j, כולל האמצעים והקוד החיצוני שהשתמשתי בו לצורך המימוש
5. את תוצאות ההרצה על בסיס נתונים סינטטי גדול שייצרתי, והמסקנות הנובעות מהן בראי מטרת הפרוייקט

# מטרת הפרוייקט

פרויקט זה עוסק בחקר ביצועים של בסיסי נתונים גרפיים. מטרת הפרויקט היא השוואת ביצועים של בסיס נתונים מסחרי בשם Neo4J לעומת שימוש באינדקס הנקרא:

.Gindex לשם הפרוייקט הנ"ל, ברצוננו לממש פתרון לבעיית התאמת התבניות מעל בסיס נתונים מסוג Neo4j. למרבה ההפתעה, למרות שבסיס נתונים זה נפוץ בשימוש מסחרי בעולם, לא קיים פתרון מובנה לבעיה זו, אלא רק פתרון מובנה לשאילתות מסלול.

התזה אתה אני יוצא לדרך, ואותה היה ברצוני להוכיח, הינה כדלקמן-

בסיס נתונים Neo4j ייתן ביצועים טובים יותר מ- gIndex על שאילתות מסלול (כיוון שזהו ייעודו מלכתחילה), אך ייתן ביצועים פחות טובים מ- gIndex עבור שאילתות גרף. כמו כן, ככל שהגרף יותר גדול, כך הפער בין השניים אמור לגדול.

ההסבר האינטואיטיבי לעניין הינו, שביצוע שאילתת גרף על Neo4j הינה פעולה מסובכת, ודורשת ביצוע הרבה שאילתות )Cypher זו שפת השאילתות של Neo4J , ) ולכן ככל שהשאילתא שנריץ גדולה יותר, כמות שאילתות ה- Cypher תגדל גם כן, וכן ייגדל הפער בין Neo4j ל- gIndex.

# מבוא

בעולם החדש בו אנו חיים, הנוצר כתוצאה מהתעצמות השימוש באינטרנט ונגישות המידע לכל, נוצרו אתגרים חדשים במיצויו:

1. כמות המידע גדלה אקספוננציאלית מדי שנה ומפוזרת ברחבי האינטרנט
2. בהרבה מהמקרים ההסתכלות על המידע הינה מבחינת הקשרים בין ישויות שונות, והמידע ה"רכוב" על הקשרים הנ"ל, כאשר הישויות עצמן מעניינות פחות מהקשרים ביניהן
3. הרבה פעמים סכימות המידע תהיינה דינאמיות ומשתנות לעיתים תכופות, תוך כדי זרימת המידע ברשת, וזוהי גמישות שאינה נתמכת ב- RDBMS הקלאסיים
4. ועוד ועוד

במקרים רבים, המידע המעניין אותנו יהיה "קשרי", כלומר נתעניין לא רק בפריטי המידע עצמם, אלא בקשרים הנוצרים ביניהם (דוגמא: רשתות חברתיות). כיוון שכך, נוח לייצג את המידע בגרף- אוסף של קדקדים, שמסמלים את הפריטים, וקשתות, שמסמלות את הקשרים ביניהם. יתירה מכך, הגרפים הללו עלולים להיות עצומים בגודלם- כמות הפריטים יכולה להיות גדולה, וכמות הקשרים עלולה להיות ריבועית לכמות הפריטים.

בסיסי נתונים רלציוניים הינם קשיחים מדי, ולא תואמים את פרדיגמות העולם החדש.

לכן נוצרו בסיסי נתונים גרפיים, הבנויים בצורה המתאימה לאחסון ועיבוד גרפים, גם גרפים עצומים בגודלם.

אחת הבעיות הראשיות בסוג חדש זה של בסיסי נתונים, היא בעיית החיפוש היעיל.

בעיה נוספת הקשורה אליה, ומוכרת מהעולם הישן (בסיסי נתונים סטנדרטיים)- מציאת קבוצת פריטים שכיחים בקבוצת טרנזאקציות. כאן, כיוון שאנחנו מדברים על עולם חדש- עולם בסיסי הנתונים הגרפיים, המאחסנים גרפים עצומים ועתירי מידע- ניאלץ לפתח שיטות חדשות להתמודדות עם אותה בעיה לוגית. בעולם הישן השתמשנו בשיטות כמו FP-Growth ו- Apriori וניאלץ להתאים אותן לדומיין החדש עליו אנחנו מדברים, ולכל סוגי המידע האפשריים- מידע מדעי, מרחבי, רלציוני וכו'.

בכדי ליישם את החיפוש היעיל, כמו בעולם הישן, נוכל לבנות אינדקס מתאים לבסיס הנתונים הגרפי. בעבודה המסכמת הקשורה לפרויקט זה אנו מסכמים מספר מאמרים וגישות לבניית אינדקסים שכאלה. כאן רק נסקור אותם בקצרה.

## בסיסי נתונים גרפיים

### מהו בסיס נתונים גרפי?

גרף הינו מודל מידע המורכב מצמתים, אשר מחוברים ביניהם בקשתות. כל צומת מייצג ישות, וכל קשת מייצגת קשר בין שתי ישויות (צמתים).

בסיס נתונים גרפי מאפשר פעולות CRUD (Create, Read, Update ו-Delete) מעל מודל נתונים מסוג גרף.

מכאן, שהדגש בבסיס נתונים מסוג זה הוא על הקשרים ולא על הישויות עצמן, בשונה מסוגי בסיסי נתונים אחרים, גם RDBMS וגם NOSQL. לכן, אחסון, אחזור ומיצוי מידע על הקשרים הנ"ל הינם פשוטים הרבה יותר מכל בסיס נתונים אחר- זאת, כיוון שמודל המידע מותאם לכך במיוחד.

קיימים שלושה מאפיינים ראשיים של בסיסי נתונים גרפיים:

1. אחסון ייעודי

בבסיס נתונים המתמחה במודל נתונים גרפי צריך לדעת לשמור אותו בצורה ייעודית, בכדי למקסם ביצועית את השליפות והפעולות שאפשר לבצע מעליו.

1. מנוע עיבוד גרפי

בכדי לבצע פעולות CRUD בצורה יעילה, נדרש מנוע עיבוד ייעודי, שיידע לעבוד עם האחסון הייעודי של בסיס הנתונים וינצל את יכולותיו בצורה מלאה.

1. שפה ייעודית

לחלק מ בסיסי הנתונים הגרפיים קיימות שפות שאילתות.

לדוגמא: ל- Neo4J שפת שאילתות ייעודית בשם Cypher, הנותנת אפשרות לבצע שאילתות ופעולות מורכבות מעל בסיס הנתונים.

## הגדרה פורמלית של הבעיה

### עיבוד שאילתא גרפית (pattern matching)

בהינתן מסד נתונים גרפי D, ושאילתא q שהיא גרף בעצמה, מצא את כל הגרפים כך, שהשאילתא q היא תת-גרף בכל אחד מהם. השאילתא q יכולה להיות כל משפט sql-י, בהתייחס לגרפים.

### האלגוריתם הנאיבי

1. עבור שאילתא q, סרוק את כל בסיס הנתונים D
   1. עבור כל גרף , בדוק האם q איזומורפית לתת גרף של g
   2. אם כן, הוסף אותה ל-
2. החזר את

#### בעיה

האלגוריתם, כמובן, לא יעיל- איזומורפיזם הינו פעולה יקרה (NPC), ולבצע אותו עבור כל הגרפים בבסיס הנתונים, שעלול להיות עצום מימדים, היא לא פעולה ישימה. כמו כן, השאילתא עצמה יכולה להיות מורכבת מקשתות רבות, מה שהופך את התהליך ליקר עוד יותר.

#### הפתרון: אינדקסים

נרצה לייעל את תהליך התשאול ע"י בניית אינדקסים מתאימים, ולהריץ את השאילתא בעזרת האינדקס, ולא רק על המידע הגולמי.

##### בעיה

לא ניתן לבנות אינדקס על סמך קשתות וקודקודים, כיוון שקשתות וקודקודים לא מזהים גרף באופן ייחודי. מצד שני, אוסף תת גרפים יכול לזהות גרף באופן ייחודי, אך בגלל מספרם הגדול, האינדקסים יתפחו למידות שהופכות אותם לבלתי אפשריים לאחסון ולתחזוקה. בכדי לפתור את הבעיה הנ"ל, נבחן מספר גישות שונות.

ישנן מספר גישות לאינדוקס גרפים:

### גישת אינדוקס המסלולים\*

הגדרה: מסלול

רצף קודקודים כך ש-

הגדרה: שאילתת XML

שאילתת XML היא שאילתת גרף הבנויה מביטויים המייצגים מסלולים בגרף.

יש שיטות אינדוקס רבות המבוססות על שאילתות XML, ומסלול הוא יחידת האינדוקס הבסיסית בכולן.

#### הרעיון הכללי

נתאר את התהליך בכללותו-

1. נקבע maxL
2. נעבור על כלל הגרפים בבסיס הנתונים
3. עבור כל גרף g נמפה את כל המסלולים שבו עד אורך maxL, ונכניס אותם לאינדקס
4. בהינתן שאילתא q
   1. נמפה את כל המסלולים ב- q עד אורך maxL
   2. נשווה אותם מול האינדקס שבנינו, ונמצא את כל הגרפים שמכילים את המסלולים הנ"ל

יתרונות

1. עיבוד מסלולים הרבה יותר פשוט מעיבוד גרפים ועצים
2. גודל האינדקס ידוע מראש

חסרונות

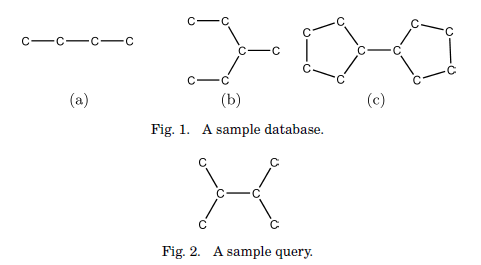
1. כאשר שוברים את הגרפים והשאילתות, והופכים אותם לאוסף מסלולים, אנו מאבדים מידע חשוב על המבנה שלהם- השלם גדול מסכום חלקיו.

מכאן, אנו עלולים לקבל false positives רבים.

1. בבסיסי נתונים גדולים אנו עלולים לקבל כמות עצומה של מסלולים.

כלומר, התהליך שתיארנו אינו יעיל.

דוגמא:



בגישה שתיארנו, בה נמפה את כלל המסלולים בשאילתא ובגרפים, התשובה תחזיר לא רק את הגרף c, שהוא התשובה לה היינו מצפים, אלא גם את הגרפים ב-a וב- b, כיוון שהם כולם מכילים את כלל המסלולים המוכלים בשאילתא.

כמו שנראה מייד, קיימת אלטרנטיבה לגישה מבוססת המסלולים.

### גישת אינדוקס תת-גרפים

\* [1][2] [3][4]

כמו שראינו בסעיף הקודם, גישה מבוססת מסלולים במלואה אינה מתאימה לאינדוקס גרפים. אם כך, נשאלת השאלה- מהי יחידת הבניין הקטנה ביותר של גרף, אשר אינה קודקוד, אינה קשת ואינה מסלול? התשובה הברורה היא- גרף. כלומר, נרצה לבנות אינדקס מבוסס תתי-גרפים של הגרפים בבסיס הנתונים. הבעיה כאן, היא שמספר תת הגרפים יכול להיות אקספוננציאלי ביחס למספר הגרפים בבסיס הנתונים- שזה מספר עצום בבסיסי נתונים גדולים. ישנן כמה גישות שהופכות אינדוקס מבוסס גרפים לישים.

### גישת אינדוקס תת-עצים

[3]דומה לגישת אינדוקס תת-הגרפים, אך יכולת הניפוי בה יותר נמוכה ככלל, בגלל שתת-עצים מכילים פחות מידע "מבני" מתת-גרפים.

מאידך, משאבי הזמן והמקום הנדרשים לאינדוקס בשיטה זו הם נמוכים יותר מגישת תת-הגרפים.

## מינוחים והגדרות

### גרף לא מכוון עם תוויות

גרף הינו קבוצת קדקדים וקשתות, מסומנים כ- . הינה קבוצת התוויות האפשריות על קשתות וקדקדי הגרף, ו הינה פונקציית התווית. כלל הגרפים בעבודה זאת הינם גרפים לא מכוונים עם תוויות. גודל גרף מוגדר להיות כמות הקדקדים שלו. תווית יכולה להיות על צומת או על קשת.

### תת-גרף

גרף הינו תת-גרף של אם וקיים איזומורפיזם תת-גרפים מ-g ל-g'. במקרה זה נגיד ש g מכיל את g' ו-g' מוכל ב-g.

### גרף-על

גרף הינו גרף-על של אם וקיים איזומורפיזם תת-גרפים מ-g' ל-g. במקרה זה נגיד ש g' מכיל את g ו-g מוכל ב-g'.

### איזומורפיזם תת-גרפים

בהינתן שני גרפים נאמר ש- G איזומורפי ל-G' אם קיים מיפוי חד-חד ערכי כך ש-

1. לכל מתקיים
2. לכל ב- כאשר מתקיים
3. לכל מתקיים

המיפוי f מוגדר כאיזומורפיזם תת-הגרפים בין G ל- G'.

### ייצוג קנוני/תווית קנונית של גרף

ייצוג קנוני של גרף הינו מחרוזת המתארת את הגרף באופן ייחודי, כך ששני גרפים בעלי אותו ייצוג קנוני יהיו בהכרח איזומורפים האחד לשני. קיימות מספר דרכים לחשב ייצוג קנוני. ניתן דרך מקובלת אחת לבצע חישוב ייצוג קנוני- עבור גרף G נגדיר מטריצת שכנויות M, אשר כל עמודה וכל שורה בה מתייחסת לקדקוד בגרף, ומסומנת בתווית של הקדקוד הנ"ל. נסמן בערך 0 אם אין קשת בין הקדקוד של השורה לקדקוד של העמודה, ואם יש קשת, נסמן את התא בתווית של הקשת (1 אם אין תווית כזו).

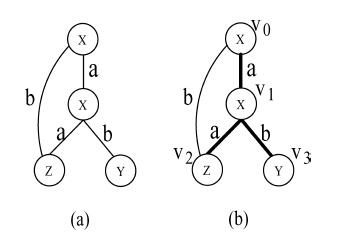
* + 1. מציאת תווית קנונית- קידוד DFS ראה [5]
       1. *האלגוריתם*

התהליך:

1. נסרוק את הגרף ב- DFS
2. כל קשת, כאשר נמצא אותה, תישמר ברשימה ממויינת (קשתות אלו נקראות הקשתות הקדמיות)
3. נסרוק את כל הקודקודים, ונכניס לאותה הרשימה את כל הקשתות שלא גילינו מקודם (אלו נראות הקשתות האחוריות)

הרשימה שקיבלנו, בה כל קשת מיוצגת כ- *, נקראת קידוד DFS. כמובן, שקיימים מסלולים רבים בהם DFS יכול להתקדם, לכן נבחר את האחד עם התווית הקטנה ביותר.*

*דוגמא:*

**

*עבור הגרף ב-a הסריקה תניב את התווית הקנונית .*

*החשיבות של הייצוג הקנוני היא שכדי לבדוק איזומורפיזם של שני תתי גרפים מספיק לבדוק שהייצוג הקנוני שלהם זהה [5].*

## תיאור האלגוריתם עצמו - gIndex

באינדקס זה משתמשים בתתי גרפים שכיחים. ראשית עוברים על כל בסיס הנתונים ומוצאים את קבוצת כל תתי הגרפים השכיחים. מתוך קבוצה זו מסננים תתי גרפים שלא מוסיפים מידע ומשאירים רק תתי גרפים שהם Discriminative

הגדרה: איבר מיותר

איבר x הוא מיותר בקבוצה F אם הוא קרוב ל-

*הגדרה: Discriminative fragment*

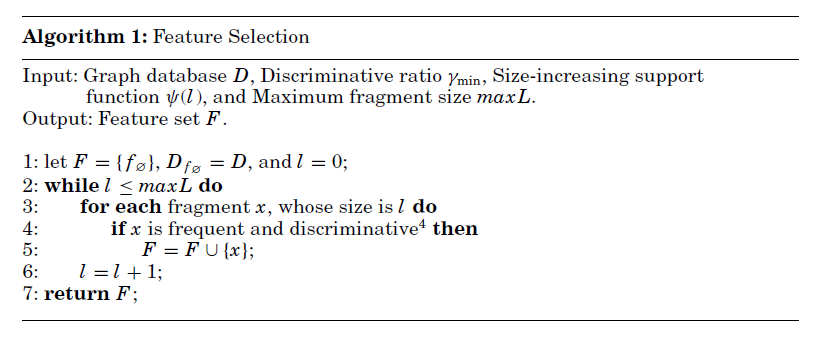
איבר x הוא Discriminative בקבוצה F אם הוא קטן מ-

*כל X הוא או מיותר או Discriminative.*

### מציאת קבוצת פריטים discriminative שכיחים

התהליך:

1. עבור כל רמה
   1. נמצא את קבוצת הפריטים השכיחים
   2. נגזום את אלו שהם לא discriminative
   3. נוסיף את מה שנשאר ל-F



כעת, משיש בידינו את קבות האיברים השכיחים וה- discriminative, נבנה את האינדקס:

1. נהפוך אותם למחרוזות (כל גרף מומר למחרוזת ע"י מציאת קידוד ה-DFS המינימלי שלו)
2. נבנה מהמחרוזות prefix tree
3. לכל איבר נשמור את רשימת המזהים של הגרפים שמכילים אותו

### העץ gIndex tree

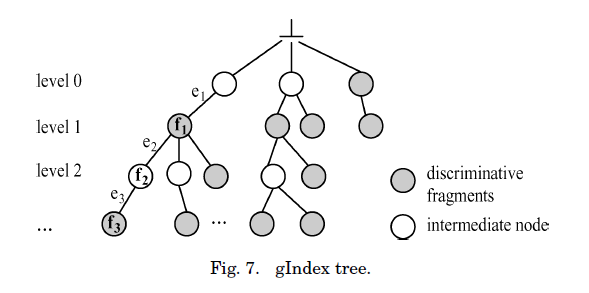
#### תיאור

עבור כל איבר, נמצא את קידוד ה- DFS המינימלי שלו, ונכניס את הקידודים של כולם ל- prefix tree, הרי הוא ה- gIndex tree. בעץ שייצרנו עלולים להיות גם צמתים מיותרים, הנמצאים שם רק כדי לחבר את העץ (צמתי ביניים). שורש העץ הוא .

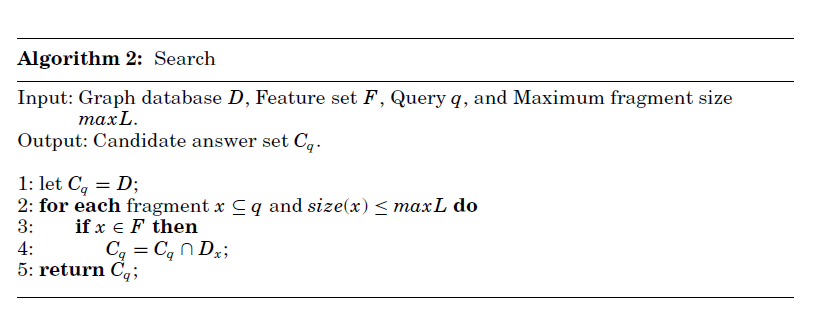
*בכל צומת, המייצג את האיבר נשמור את רשימת המזהים של הגרפים שמכילים אותו .*

לכן, אם נרצה לדעת מה כל הגרפים שמכילים גם את וגם את , הרי שנצטרך לבצע פשוט .

*דוגמא:*

*בדוגמא זו ו- .* שניהם קידודי DFS של איברים שכיחים ו- discriminative.מצד שני,  *הוא לא discriminative.*

### שלב החיפוש

בגדול:

1. עבור על כל האיברים בשאילתא
2. מצא כל איבר באינדקס
3. מצא את כל רשימות המזהים של כל איבר בדיסק
4. חתוך את כל רשימות המזהים בכדי לקבל רשימת מועמדים

### שלב האימות

השלב הטריוויאלי- בהינתן קבוצת מועמדים , פשוט נבצע איזומורפיזם על כל הגרפים בבסיס הנתונים, בכדי לאמת את המועמדים.

## מהו Neo4j

Neo4J הינו בסיס נתונים הגרפי המוביל היום בשוק. הוא פותח ע"י חברת Neo Technologies ויצא לשוק בשנת 2007. בסיס הנתונים הנ"ל הינו בתקן קוד פתוח ופותח בשפת Java.

בסיס הנתונים הינו נטול סכימה- כלומר, הוא מאפשר לנו גמישות מקסימלית באחסון המידע, כיוון שהמידע המאוחסן לא מוגבל בסכימה.

### מאפייני ה-ACID

בסיס הנתונים מממש תקן ACID (Atomicity, Consistency, Isolation, Durability) על טרנזאקציות-

מאפיין Atomicity

אם חלק מטרנזאקציה נופל, אזי הנ"ל מפיל את הטרנזארציה כולה. כלומר- הטרנזאקציה, אף שיכולה להיות מורכבת ממספר חלקים רב, תתחרש כפעולה אטומית.

מאפיין Consistency

הטרנזאקציה מובילה ממצב ולידי של בסיס התונים למצב ולידי אחר. כלומר, כל התנאים וההגבלות החלים על הבסיס והמידע המאוחסן בו עדיין בתוקף.

מאפיין Isolation

אם שתי טרנזאקציות רצות במקביל, תוצאת ריצתן תהייה זהה למקרה בו היו רצות בצורה סדרתית.

מאפיין Durability

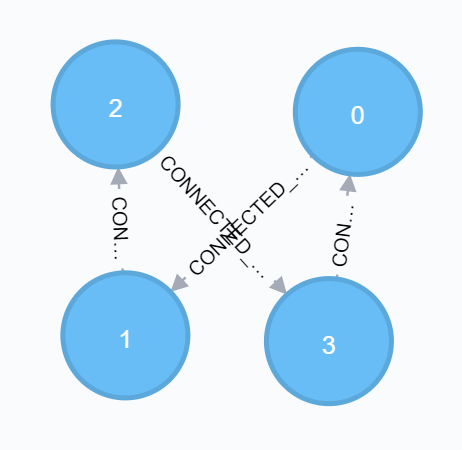
אם בסוף הטרנזאקציה התבצעה פעולת Commit, כלומר, הטרנזאקציה הצליחה- תוצאתה תישמר, גם במקרה וקרתה לאחר מכן שגיאה, נפילה או כל תקלה אחרת.

### שפה ייעודית

ל- Neo4J שפת שאילתות ייעודית בשם Cypher, הנותנת אפשרות לבצע שאילתות ופעולות מורכבות מעל בסיס הנתונים.

דוגמא:

נניח וקיים לנו בבסיס הנתונים הגרף הבא בלבד:



בגרף הנתון לכל צומת יש מזהה ותווית זהות, וערכן מצויין על גבי הצומת. גם על קשת יש תווית.

ניתן דוגמא לשתי שאילתות שונות ולתוצאותיהן:

אם נרצה להחזיר את הצומת עם מזהה 0-

match (n:Node { id: 0 }) return n

נקבל-



אם נרצה להחזיר את כל הקשתות-

match (n:Node)-[r:CONNECTED\_TO]-(m:Node) return r

נקבל-

╒═════════════════════════════╕

│"r" │

╞═════════════════════════════╡

│{"label":2}│

├─────────────────────────────┤

│{"label":3}│

├─────────────────────────────┤

│{"label":0}│

├─────────────────────────────┤

│{"label":1}│

├─────────────────────────────┤

│{"label":1}│

├─────────────────────────────┤

│{"label":2}│

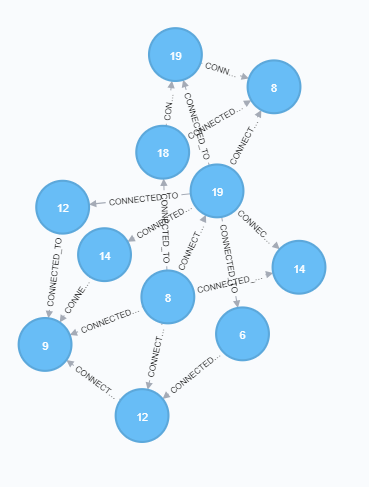
├─────────────────────────────┤

│{"label":0}│

├─────────────────────────────┤

│{"label":3}│

יהיה הגרף הבא, בעל מזהה 1, בבסיס הנתונים-



נצפה שהשאילתא הבא תחזיר אותו וכל גרף אחר שעונה עליה-

match (n:Node {label : 19}) - [:CONNECTED\_TO {label:1}] - (m:Node {label: 8}) - [:CONNECTED\_TO {label : 1}] - (:Node {label:19}) return distinct (n.graphId)

הסבר- השאילתא הנ"ל מחזירה את המזהים של כל הגרפים שקיים בהם צומת בעל תווית 19, המחובר לצומת בעל תווית 8 דרך קשת בעלת תווית 1, והצומת בעל התווית 8 בתורו מחובר לצומת בעל תווית 19 דרך קשת בעלת התווית 1.

בבסיס הנתונים עליו עשיתי את הניסויים בעבודה זו, השאילתא הנ"ל תחזיר 736 מזהים, וביניהם את המזהה 1.

## מבנה העבודה

### בעיית התאמת התבניות ב- Neo4j

שפת השאילתות של Neo4j, הידועה בשם Cypher, מאפשרת לבצע שאילתות מסלול (Path), ואין בה אפשרות לבצע בצורה מובנית שאילתות גרפים (כלומר, התאמת תבניות). לכן, השלב הבסיסי הינו לממש התאמת תבניות ב- Neo4j. לצורך זאת כל שאילתא תחולק למסלולים, שיוסבו לשפת Cypher, ותוצאותיהן יאוחדו ויאומתו ע"י איזומורפיזם.

### מימוש gIndex [4]

לצורך ההשוואה אל מול ביצועי ה- Neo4j בבעיית התאמת התבניות נבחר אלגוריתם gIndex. האלגוריתם הנ"ל היווה פריצת דרך בעת יציאתו, ואל מולו נערכות רוב ההשוואות של האלגוריתמים שבאו עכשיו. יתירה מכך, בכדי לבטל את הבדלי הביצועים בשליפות/שמירות/אחסון, מימוש ה- gIndex בוצע גם הוא מעל Neo4j.

### בסיס נתונים לבדיקה

צורך הבדיקות הנ"ל, כדי לקבל תוצאות אינדיקטיביות, נדרש בסיס נתונים גדול יחסית. בכדי לייצר בסיס נתונים שכזה השתמשתי בכלי GraphGen [6] שעל בסיס פלטיו נעשו ההשוואות גם ב-[2][1]. כלי זה יודע לבנות בסיסי נתונים סינטטיים, בגדלים משתנים ועם פרמטרים משתנים, לפי דרישה.

בסיס הנתונים שייצרתי לצורך הרצת הבדיקות מורכב מ-10,000 גרפים, בגודל ממוצע של 20 קשתות לגרף.

## סביבת העבודה

הבדיקות התבצעו במחשב עם מעבד i7, עם 16 גיגהבייט זכרון RAM. בסיס הנתונים Neo4j הנבדק היה מגרסת 3.2.3. הקוד נכתב בסביבת .Net 4.5

## התזה

התזה אתה יצאתי לדרך, ואותה היה ברצוני להוכיח, הינה כדלקמן-

בסיס נתונים Neo4j ייתן ביצועים טובים יותר מ- gIndex על שאילתות מסלול (כיוון שזהו ייעודו מלכתחילה), איך ייתן ביצועים פחות טובים מ- gIndex עבור שאילתות גרף. כמו כן, ככל שהגרף יותר גדול, כך הפער בין השניים אמור לגדול.

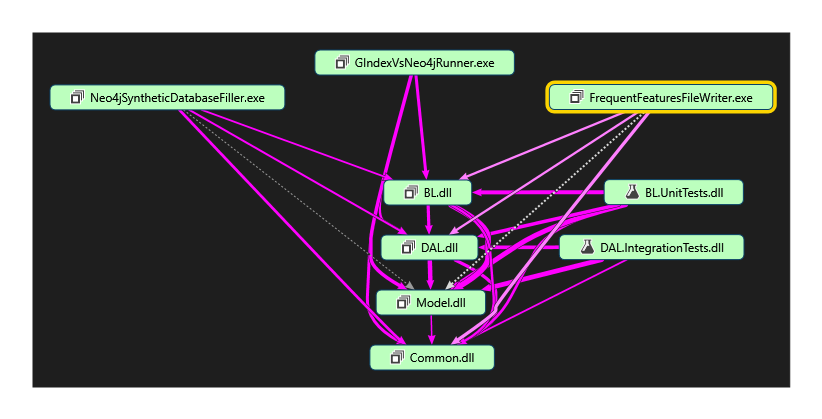
ההסבר האינטואיטיבי לעניין הינו, שביצוע שאילתת גרף על Neo4j הינה פעולה מסובכת, ודורשת ביצוע הרבה שאילתות Cypher, ולכן, ככל שהשאילתא שנריץ גדולה יותר, כמות שאילתות ה- Cypher תגדל גם כן, וכן ייגדל הפער בין Neo4j ל- gIndex.

# מימוש הפרוייקט

## תיאור ומבנה כללי

המימוש הינו קוד C# בסביבת .Net 4.5, והוא משתמש ב- Neo4j מגרסא 3.2.3.

המימוש מחולק לפרוייקטים לפי מודל השכבות-



נבחן כל אחד מהפרוייקטים הנ"ל בנפרד, ונראה את הקשרים ביניהם.

הערות:

לצורך פשטות עשיתי מספר הנחות על המימוש של gIndex המתואר בסעיף הקודם ובמאמר [4]-

1. במקור לוקחים מתוך קבוצת ה- Frequent Features רק את תת קבוצת ה- Discriminative Features מביניהם, אך אני לא עשיתי את הסינון הנ"ל.
2. במקור מתואר מימוש ייחודי של prefix tree (gIndex tree), הממומש ע"י טבלת גיבוב ומאפשר ביצוע פעולות ניפוי אפריורי והקטנת מספר לעולות חיתוך רשימות המזהים ע"י שימוש באיברים discriminative מקסימליים (אם בשאילתא ישנם שני איברים ,  *הרי שרשימת המועמדים של מכילה את רשימת המועמדים של ).* במקום מימוש זה אני השתמשתי במימוש קיים מספריית Nuget (ספריית שיתוף המימושים של Microsoft) בשם VDS.Common [8].

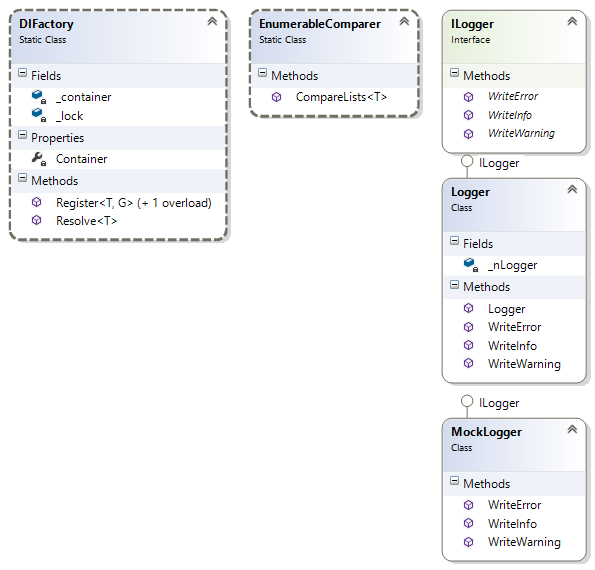
בהנחה ואני מוכיח את מטרת הפרוייקט בהתחשב בהנחות הללו, אז כל שכן אם אני מיישם את היוריסטיקות והשיפורים של המימוש המקורי, ההוכחה תהייה תקיפה באופן עקרוני, וכל שיפור ויוריסטיקה נוספים רק יגדילו את הפער.

* + 1. האלגוריתמים שמימשתי

מבין האלגוריתמים העיקריים חלקם מימשתי וחלקם נלקחו מהרשת בפירוט:

* + - 1. מציאת תתי גרפים שכיחים – נלקח מהרשת [7]
      2. בדיקת איזומורפיזם של תתי גרפים – נלקח מהרשת [7]
      3. בניית האינדקס – מימשתי
      4. הרצת שאילתות הן מעל האינדקס והן מעל ה- Neo4j ללא שימוש באינדקס - מימשתי
      5. שבירת שאילתא למסלולים – מימשתי

## פרוייקט Common



### מחלקה DIFactory

בשביל היכולת לכתוב טסטים לקוד, השתמשתי בספרייה Unity, שמספקת יכולת Dependency Injection. המחלקה DIFactory מייחצנת יכולות אלו לכלל הפרוייקטים.

### מחלקות ה-Logger-ים

המחלקה Logger ומייחצנת יכולות Logging מרוכזות לכלל המימוש.

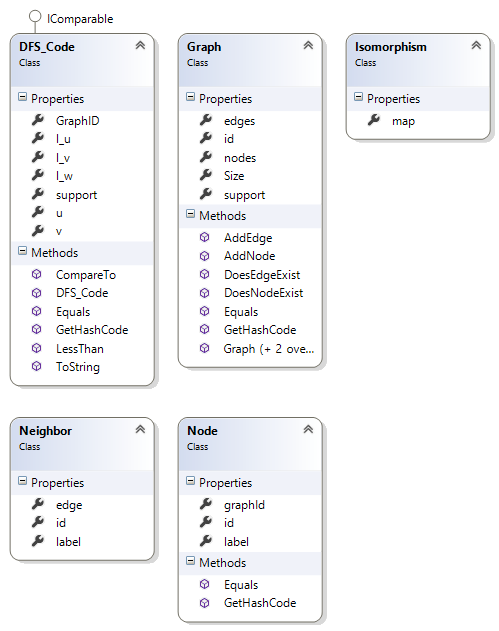
MockLogger הינה מימוש של ILogger לטסטים.

### מחלקה EnumerableComparer

מחלקה המייחצנת פונקציונליות של השוואת Lists.

## הפרוייקט Model

הפרוייקט מכיל את מודל הנתונים של כלל המימוש, וכל הפרוייקטים האחרים מסתמכים עליו.



### מחלקה Graph

מחלקה המייצגת גרף.

### מחלקה Node

מחלקה המייצגת צומת בגרף.

### מחלקה DFS\_Code

מחלקה המייצגת קשת בגרף.

### מחלקה Neighbor

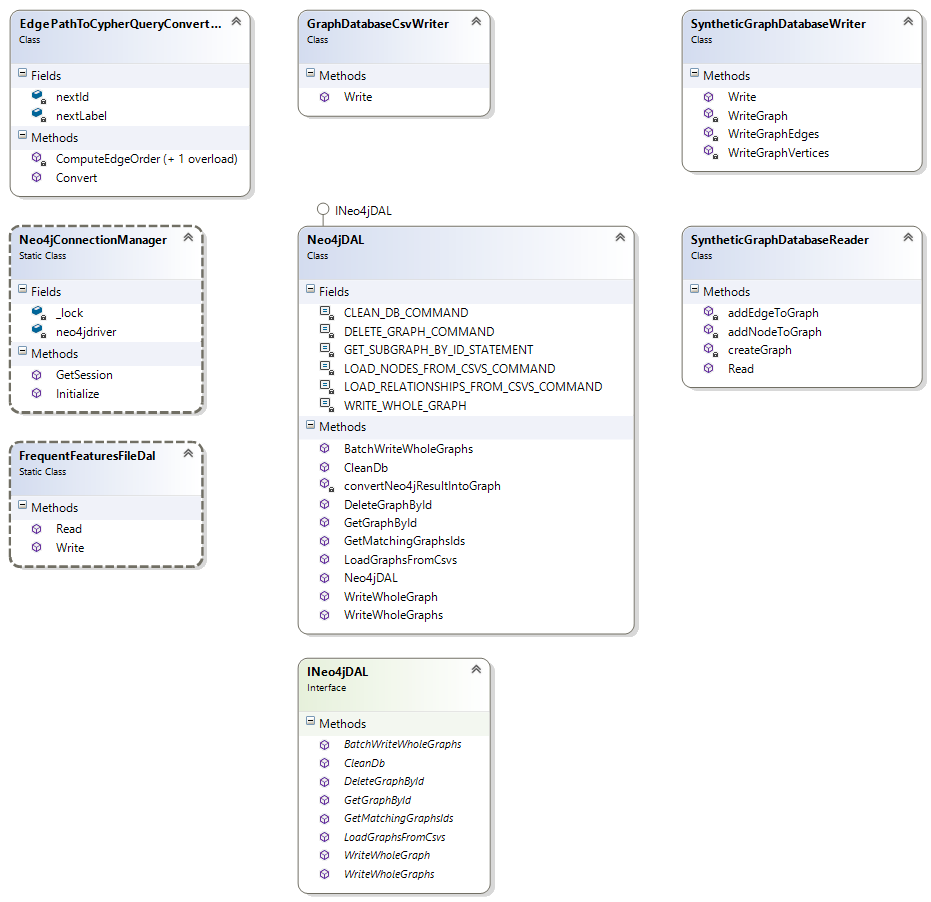
מחלקה המייצגת שכן של צומת בגרף

### מחלקה Isomorphism

מחלקה המייצגת מיפוי איזומורפיזם תת-גרפים בין שני גרפים.

## פרוייקט DAL

הפרוייקט עושה אנקפסולציה מעל התקשורת ל- Neo4j ומעל קריאת וכתיבת בסיסי נתונים גרפיים לקבצי טקסט.



### מחלקה Neo4jDAL

Neo4jDAL יורשת מהמנשק INeo4jDAL ומייחצן API המאפשר גישה ל- Neo4j.

להלן מתודות העיקריות הנורשות מהמנשק Ineo4jDAL-

#### מתודה LoadGraphsFromCsvs

מתודה היודעת למלא את בסיס הנתונים מקובץ CSV המייצג רשימת גרפים.

#### מתודה DeleteGraphById

מחיקת גרף לפי מזהה.

#### מתודה CleanDb

מחיקת כלל הגרפים בבסיס הנתונים.

#### מתודה GetMatchingGraphsIds

1. המתודה מקבלת כקלט רשימת קשתות DFS\_Code וממירה אותה לשאילתת מסלול בשפת Cypher בעזרת המחלקה EdgePathToCypherQueryConverter
2. מריצה את השאילתא על בסיס הנתונים, וממירה את התשובה לרשימת מזהי גרפים
3. מחזירה את רשימת המזהים

#### מתודה GetGraphById

מחזירה גרף שלם מבסיס הנתונים על סמך המזהה שלו.

### מחלקות SyntheticGraphDatabaseReader ו- SyntheticGraphDatabaseWriter

מייחצנות יכולת קריאה וכתיבה של בסיסי נתונים גרפיים המאוחסנים בקבצים, בהתאמה.

### מחלקה FrequentFeaturesFileDal

אחראית על כתיבת ה- FF לקובץ, לשם שימוש חוזר (זמן הכרייה עלול להיות מאוד ארוך, ולכן יש ייתרון בלשמור אותן לשימוש חוזר).

### מחלקה GraphDatabaseCsvWriter

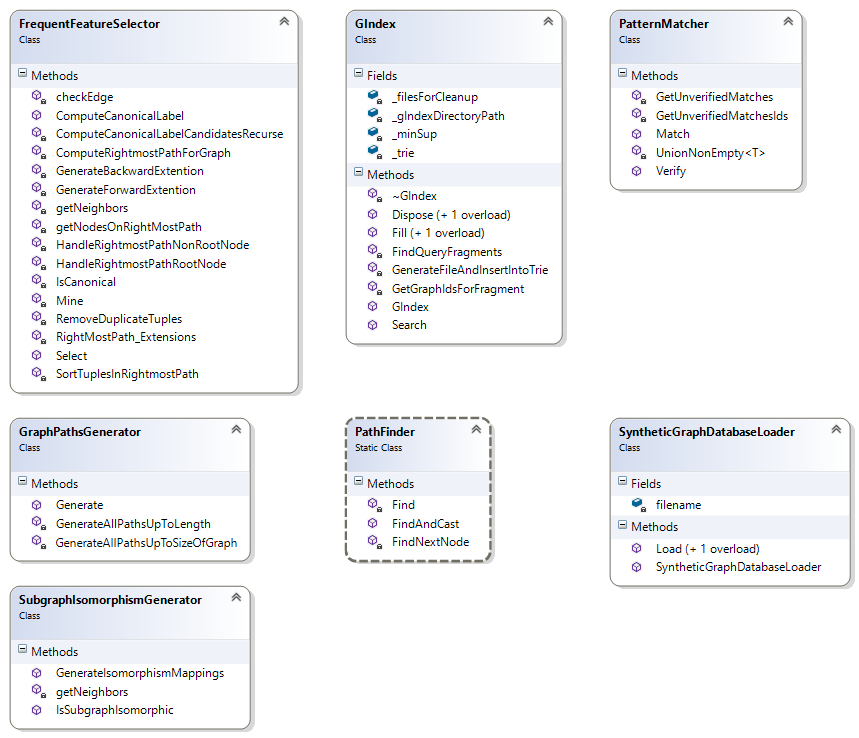
יודעת לטעון רשימת גרפים לקובץ CSV, לשם טעינתם ל-Neo4j בצורה יעילה.

### מחלקה EdgePathToCypherQueryConverter

יודעת להמיר רשימה של קשתות DFS\_Code לשאילתת Cypher.

## פרוייקט BL

מכיל את כלל הלוגיקה של המימוש.



### מחלקה FrequentFeatureSelector

מייחצנת פונקציונליות של מציאת FrequentFeatures מבסיס נתונים גרפי. היא עושה זאת לפי האלגוריתם המתואר במאמרים [4][5]. עשיתי שימוש במימוש מאת כותבי מאמר [7] לאלגוריתם gSpan.

### מחלקה PathFinder

מחלקה היודעת, באופן רקורסיבי, למצוא את כלל המסלולים בגרף, עד לגודל מסויים שהיא מקבלת כקלט.

### מחלקה GraphPathsGenerator

מייחצנת פונקציולניות של מציאת כלל המסלולים בגרף מסויים, בצורה רקורסיבית. היא עושה שימוש במחלקה PathFinder.

### מחלקה SubgraphIsomorphismGenerator

SubgraphIsomorphismGenerator יודעת למצוא האם גרף הינו תת-גרף איזומורפי לגרף אחר, ואם כן, מה המיפוי התואם את האיזומורפיזם הנ"ל. גם כאן המימוש הינו תואם את המתואר במאמרים [4][5].

### מחלקה SyntheticGraphDatabaseLoader

SyntheticGraphDatabaseLoader הינה מחלקה היודעת לטעון בסיס נתונים גרפי מקובץ ישירות ל- Neo4j.

התהליך-

1. מקבלת כקלט של קובץ
2. טוענת אותו בעזרת SyntheticGraphDatabaseReader מפרוייקט ה- DAL
3. הופכת את רשימת הגרפים לקובץ CSV בעזרת GraphDatabaseCsvWriter מפרוייקט ה- DAL
4. טוענת את ה-Neo4j מה- CSV

### מחלקה PatternMatcher

PatternMatcher הינה מחלקה היודעת לבצע התאמת תבניות מעל Neo4j בצורה נאיבית.

GIndex הינה מימוש אלגוריתם gIndex.

התהליך-

1. עבור שאילתא q
2. מוצאת את כלל המסלולים ב- q בעזרת GraphPathsGenerator מה- DAL
3. עבור כל מסלול מריצה שאילתא ע"י ה- Neo4jDAL
4. מאחדת את כלל התוצאות לרשימת IDים אחת
5. שולפת לפי כל ID את הגרף שלו בעזרת ה- Neo4jDAL
6. מאמתת את השאילתא אל מול המועמדים בעזרת איזומורפיזם (SubgraphIsomorphismGenerator)

## פרוייקט Neo4jSyntheticDatabaseFiller

הפרוייקט מקבל כקלט קובץ שמייצג בסיס נתונים גרפי ויודע לטעון אותו ל- Neo4j.

התהליך-

1. מקבל כקלט מסלול לקובץ המייצג בסיס נתונים גרפי
2. מנקה את Neo4j
3. מפעיל את SyntheticGraphDatabaseLoader מפרוייקט ה- BL וטוען את בסיס נתונים ל- Neo4j

## פרוייקט FrequentFeaturesFileWriter

הפרוייקט מקבל כקלט קובץ שמייצג בסיס נתונים גרפי וסף minSup, מחשב את ה- FrequentFeatures, ושומר אותם לקובץ לשימוש מאוחר יותר.

התהליך-

1. מקבל כפרמטרים שם קובץ שמייצג בסיס נתונים גרפי, minSup מינימלי, minSup מקסימלי, וגודל קפיצה
2. טוען את בסיס הנתונים מהקובץ
3. מאתחל את i להיות ה- minSup המקסימלי
4. בלולאה
   1. אם i שווה למינימום- עוצר
   2. כורה את ה- FrequentFeatures מבסיס הנתונים עבור minSup=i
   3. שומר אותם לקובץ
   4. מקטין את i בגודל הקפיצה

## פרוייקט GIndexVsNeo4jRunner

הפרוייקט הראשי, מקבל קובץ שמייצג בסיס נתונים גרפי, קובץ שמייצג FrequentFeatures, את הסף minSup של הקובץ הקודם וקובץ שמכיל את גרפי השאילתות.

התהליך-

1. מקבל כפרמטרים מסלול לקובץ שמייצג את בסיס הנתונים הגרפי, מסלול לקובץ ה- FrequentFeatures, מסלול לקובץ השאילתות, ואת ה- minSup שאיתו נכרו ה- FrequentFeatures
2. מרוקן את ה- Neo4j
3. טוען את הגרפים מבסיס הנתונים ל- Neo4j
4. בונה אובייקט GIndex מבסיס הנתונים
5. עבור כל שאילתא ברשימת השאילתות
   1. מבצע התאמת תבניות מול ה- Neo4j ישירות
   2. מבצע התאמת תבניות בעזרת ה- GIndex וכאשר ה- GIndex מחזיר מזהים שולף אותם מה- Neo4j
   3. כותב את התוצאה לקובץ CSV

# תוצאות

דבר ראשון התנסיתי בזמני בניית ה-gIndex ושינויים ע"י קביעת minSup שונה. חלק הארי של זמן בניית האינדקס (השאר זניחים) הינו כריית הפריטים השכיחים. להלן טבלה של זמני כריית FrequentFeatures מבסיס הנתונים שהרצתי עליו את הניסויים בשעות כפונקציה של minSup-

ניתן לראות כי ככל שה-minSup נמוך יותר, הרי שזמן הבנייה וכמות הפריטים תגדל. לכן, מחד עלינו לבחור minSup לא קטן מדי, ולא גדול מדי מאידך, שהרי שבאיזשהו שלב אין בידינו מספיק פריטים שכיחים בגדלים משתנים בכדי לבצע את כלל השאילתות.

להלן מספר דוגמאות לתוצאות שהתקבלו על שאילתות שהורצו-

|  |  |  |  |
| --- | --- | --- | --- |
| **זמן שליפה מ-Neo4j** | **זמן שליפה מ-gIndex** | **גודל שאילתא** | **אחוז שיפור** |
| 01:04.8 | 00:05.6 | 7 | 91.41358025 |
| 01:16.3 | 00:07.5 | 7 | 90.22223387 |
| 00:58.0 | 00:05.8 | 7 | 89.93858257 |
| 01:00.8 | 00:06.3 | 7 | 89.63986186 |
| 01:08.2 | 00:07.6 | 7 | 88.81280882 |
| 00:45.6 | 00:05.1 | 7 | 88.70854536 |
| 00:42.4 | 00:05.0 | 7 | 88.18582299 |
| 00:28.1 | 00:03.5 | 6 | 87.4777912 |
| 00:43.2 | 00:05.5 | 7 | 87.21234734 |

התוצאות שהתקבלו היוו מפתיעות קמעה- ברוב המוחלט של השאילתות היה ייתרון בשימוש ב- gIndex על פני Neo4j, זאת כולל שאילתות מסלול!

כלומר, רק ב-8 מתוך כ-5,900 (כ-0.1%) שאילתות ה- Neo4j נתן ביצועים יותר טובים (וגם זאת בצורה זניחה).

התזה, שגרסה כי שאילתא גדולה יותר תרוויח יותר משימוש ב- gIndex, התממשה- שאילתות גדולות יחסית (כ-7 צמתים) זכו במקרה הקיצוני לשיפור ביצועים של כ-91%!

במקרה הממוצע ה-gIndex שיפר את ביצועי השאילתות בכ-49%.

# מסקנות

אם נוסיף את ההנחה ההגיונית הבאה- "ככל שבסיס הנתונים גדל, וגודל כל גרף בו גדל, השאילתות עליו ייקחו יותר זמן"- הרי שלמימוש אינדקס יהיה ייתרון משמעותי יותר. ולכן, אם אדם/חברה צריך לעשות שימוש ב- Neo4j לצרכים מסחריים ו/או מדעיים, בכוונתו לאחסן בו כמות מידע גדולה, ולבצע עליו שאילתות (ואף רק שאילתות מסלול), הרי שיהיה ייתרון (ואף גדול, במקרה של שאילתות מורכבות) במימוש אינדקס מעליו. יתירה מכך- ככל שהתחום מתקדם, והאינדקסים הנוצרים יעילים יותר, מימוש אינדקס יעיל מעל Neo4j ייתן ייתרון גדול יותר.

# ביבליוגרפיה

1. James Cheng, Yiping Ke, and Wilfred Ng. Efficient query processing on graph databases. ACM Trans. Database Syst. 34, 1, Article 2 (April 2009), 48 pages.
2. James Cheng, Yiping Ke, Wilfred Ng, and An Lu. Fg-index: towards verification-free query processing on graph databases. In Proceedings of the 2007 ACM SIGMOD international conference on Management of data (SIGMOD '07). ACM, New York, NY, USA, 857-872.
3. Dennis Shasha, Jason T. L. Wang, and Rosalba Giugno. Algorithmics and applications of tree and graph searching. In Proceedings of the twenty-first ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems (PODS '02). ACM, New York, NY, USA, 39-52.
4. Xifeng Yan, Philip S. Yu, and Jiawei Han. Graph indexing based on discriminative frequent structure analysis. ACM Trans. Database Syst. 30, 4 (December 2005), 960-993.
5. Xifeng Yan and Jiawei Han. gSpan: Graph-Based Substructure Pattern Mining. In Proceedings of the 2002 IEEE International Conference on Data Mining (ICDM '02). IEEE Computer Society, Washington, DC, USA, 721-.
6. James Cheng, Yiping Ke, and Wilfred Ng. GraphGen: A graph synthetic generator. http://www.cse.ust.hk/graphgen/, 2006.
7. Bay Vo, Dang Nguyen and Thanh-Long Nguyen. A parallel algorithm for frequent subgraph mining. ICCSAMA 2015, Metz, France. Springer AISC, 358, 163-173. 2015.
8. https://www.nuget.org/packages/VDS.Common/1.9.0-pre0001