שפות תכנות 234319

תרגיל בית מס' 6

אביב ה'תשע"ו

List iteration, foreach vs iterator

- Measurement: average milliseconds / operation, less is better

23.659

```
public List<Integer> arrayListForeach() {
    for(Integer i : arrayList) {
    }
    return arrayList;
}
```

22.445

```
public Iterator<Integer> arrayListIterator() {
    Iterator<Integer> iterator = arrayList.iterator();
    while(iterator.hasNext()) {
        iterator.next();
    }
    return iterator;
}
```


העתקות: אסור להעתיק. מותר ומומלץ להתייעץ, אפשר להעביר רמזים, אבל חייבים לכתוב בא
עצמאי. אי־פשור העתקה נחשב כהעתקה: הצגת תרגיל בית לרבות טיוטה או שורת קוד למישהו או
מועד האיחורים להגשת התרגיל, נחשבת כהעתקה. האחריות על הגשה מקורית היא על שני חברי
כלומר, אם לא בדקת את השותף שלך, והוא העתיק, האשמה בהעתקה היא של שניכם. כל חשד י
(מתוך נהלי הקורס)

1. שאלות כלליות

1. **סיווג מע' פקודות:** קראו שוב את פרק 6, סווגו את מערכת הפקודות בשפת נים ע"פ הקריטריונים הבאים:
   1.1. פקודות אטומיות של השפה
   1.2. וריאנטים של פקודת ההשמה הנתמכים בשפה
   1.3. ה SEQUENCERS של השפה
   1.4. בנאי פקודות: מהם כל בנאי הפקודות
   1.5. קיום שלושת הבנאים העיקריים:
      1.5.1. אילו וריאנטים יש לבנאי השרשור
      1.5.2. אילו וריאנטים יש לבנאי התנאי
      1.5.3. אילו וריאנטים יש לבנאי הלולאה
   1.6. היכן נמצאת השפה על הספקטרום שבין פקודה לביטוי? כלומר, באיזו גישה היא נוקטת?

   תזכורת: כפי שהוסבר בשיעור ביום 1.6.2016, כל שפה בוחרת את הדרך שבה היא מערבת ומשלבת פקודות עם ביטויים, כאשר הגישות העיקריות הן:
   - שפות שבהן יש ניסיון להפרדה מוחלטת (פסקל),
   - שפות expression oriented כלומר שפות שכל ביטוי הוא גם פקודה אם מוסיפים בסופו ; וכל הפרוצדורות הן פוקציות שמחזירות את הטיפוס UNIT (למשל שפת C ושפת AWK, ובמידה מסוימת גם שפת Java),
   - שפות שאינן expression oriented, כמו למשל Bash
   - שפות שבהן אין פקודות כלל (Haskel), שפות ששני המושגים לא קיימים בהם (פרולוג), שפות COMMAND EXPRESSIONS שבהן כל פקודה היא ביטוי, וכל ביטוי הוא פקודה (למשל ICON, PostScript MOCK ועוד),
   - שפות כמו ML, שבהן אין בעצם פקודות. אין לולאות, אלא שמותרות פונקציות שמחזירות UNIT וקיים האופרטור ; (שדומה לאופרטור ה-, בשפת C) שמחשב את הארגומנט הראשון. מתעלם מערכו, ומחזיר את הארגומנט השני.

2. תרגיל בונוס **(תחילת לימוד למבחן):** פתרו שאלה שאין לה פתרון באתר השו"ת, או שפרו משמעותית תשובה קיימת. כלומר, צרפו תשובה שלכם המבוססת על תשובה קיימת ומשפרת אותה משמעותית. אנא סמנו UPVOTE על כל שאלה ששקלתם ועל כל תשובה ששיפרתם. צרפו קישור לעדכון שלכם. אין צורך להעלות שאלות לאתר בשלב זה. זאת תעשו, על פי הצורך, כחלק מהלימוד למבחן.

   אין להתחכם באמצעות "שימוש חוזר". כלומר, נאסר עליכם להשתמש במטרה זו, בפתרון של תרגילים אחרים בשאלון זה. כלומר אין להוסיף פתרון לשאלה קיימת, או לעדכן באופן משמעותי שאלה קיימת בשו"ת, אם היא דומה או מבוססת על שאלות המופיעות בגיליון זה.

3. **שאלות טיפוסיות למבחן:** כפי שהיה בתרגילים קודמים, בעבור כל זוג של סטודנטים בקורס מוגדרות שתי ספרות שונות בין 0-9 לפי הספרה האחרונה של מספרי הזהות. בחרו שני סעיפים לפי שתי ספרות אלו.

בכל אחד מסעיפי השאלה, עליכם לבחור שפה לבחירתכם ולכתוב תוכנית שמטרתה להפריד בין כל זוג מקרים. כל הסעיפים בנויים מהצורה: "על התוכנית להדפיס 1, אם A מתקיים, ו-0 אם B מתקיים". ניתן להניח שבדיוק אחד מבין A ו B מתקיים.

לדוגמה, אם A היא הטענה שאומרת "השפה מאפשרת overloading של פונקציות חיצוניות על ידי פונקציות פנימיות" והטענה B אומרת ש"השפה מסתירה פונקציות חיצוניות על ידי פונקציות פנימיות", אז הנה פתרון אחד אפשרי בשפת פייתון:

```
def f(a):
    return 1

def g():
    def f():
        return 0
    try:
        print f(1)
    except TypeError:
        print 0

g()
```

אם פייתון מאפשרת overloading של פונקציות חיצוניות על ידי פונקציות פנימיות, אז כאשר נקרא ל-g, היא תגלה שאין ב-scope הפנימי שלה פונקציה בשם f שמקבלת פרמטר אחד, תחפש אותה ב-scope שמעליו, תמצא ותדפיס 1 כנדרש.

אם, לעומת זאת, פייתון מסתירה פונקציות חיצוניות על ידי פונקציות פנימיות, אז ייזרק Exception שאומר שאין פונקציה f שמקבלת פרמטר אחד, והתוכנית תדפיס 0 כנדרש.

להלן המקרים:

סעיף	התכנית תדפיס 1	התכנית תדפיס 0
0	השפה משתמשת ב column wise	השפה משתמשת ב row wise
1	השפה משתמשת ב value-result	השפה משתמשת ב reference
2	השפה משתמשת ב normal-order	השפה משתמשת ב eager
3	השפה משתמשת ב normal-order	השפה משתמשת ב lazy+cache
4	השפה משתמשת ב value semantics ביחס לרשומה Date כפי שהיא מצויה בחוברת השקפים	השפה משתמשת ב reference semantics, ביחס לרשומה Date כפי שהיא מצויה בחוברת השקפים
5	השפה משתמשת ב Lexical Scoping	השפה משתמשת ב static scoping
6	השפה משתמשת ב short circuit של AND	השפה אינה משתמשת ב short circuit של AND
7	השפה משתמשת ב short circuit של OR	השפה אינה משתמשת ב short circuit של OR
8	השפה משתמשת ב short circuit של האופרטור הטרנרי (לא הפקודה): if-then-else הקרוי גם בשפת C: ?:	השפה אינה משתמשת ב short circuit של האופרטור הטרנרי (לא הפקודה): if-then-else הקרוי גם בשפת C: ?:
9	השפה מחשבת פעם אחת את טווח הלולאה בלולאות מסוג: for i:=1 to N do something	השפה מחשבת את טווח הלולאה בכל איטרציה של לולאות מסוג: for i:=1 to N do something

4. **עוד שאלות טיפוסיות למבחן:** בדומה לסעיף הקודם, היזכרו כי בעבור כל זוג של סטודנטים בקורס מוגדרות שתי ספרות שונות בין 0-9 לפי הספרה האחרונה של מספרי הזהות. בחרו שני סעיפים לפי שתי ספרות אלו. שימו לב שבכל סעיף יש שני מושגים.

0. Genenrators and coroutines
1. Normal order and generators
2. First class functions and normal order evalutation
3. Coroutines and normal order.
4. Normal order vs. Eager Order
5. Collateral evaluation and left to right evalution evalution order of arguments
6. Garbage collection and colsures
7. Static typing and dynamic typing
8. Lexical Scoping and dynamic Scoping
9. Short circuit evalution and eager evaluation

בכל אחד מהסעיפים שבחרתם:

a. הסבירו כל מושג בערך ב-100 מילים.
b. הסבירו אם המושגים זהים, בעלי חפיפה חלקית, זרים, או בלתי תלויים. (מושגים הם ב"ת אם בחירה באחד אינה כופה בחירה של האחר)
c. תנו דרך לממש מושג א' באמצעות ב', אם ניתן לעשות זאת. (הכוונה למשל לדרך שבה המתכנת חכם יכול אולי לחקות static typing תוך שימוש ב generators), תנו דרך למימוש מושג ב' באמצעות א', אם ניתן לעשות זאת. אם לא ניתן לעשות זאת, הסבירו מדוע, בקצרה או באריכות, לפי הזוג שנפל בגורלכם.

5. כתוב תכנית בשפת C שיוצרת שגיאת Dangling Reference באמצעות החזרת מצביע למשתנה מחסנית, והסבר מדוע לא ניתן לכתוב תכנית דומה בשפת פסקל.
6. **תרגיל בונוס**: כתוב תכניות מיקרוסקופיות (כמו בשאלה 2) בשפת פסקל שמייצרות את כל אחד משגיאות הערימה שניתן לייצר בפסקל. אם לא ניתן לעשות זאת עבור שגיאה מסויימת, הסבר מדוע.

בכל השאלות בפרק זה, אתם מתבקשים לחפש את המונחים בחוברת השקפים בדרופבוקס או ב Google Drive וכמובן גם בויקיפדיה.

2. איטרטורים ב Java

כתבו תכנית למציאת מספר המונית בשפת Java. התכנית תהיה בנויה על איטרטור שמחזיר סדרה אינסופית של מספרים שלמים (באיזה טיפוס של Java תשתמשו לייצוג הטבעיים?), שהם החזקות השלישיות של כל המספרים הטבעיים:

0, 1, 8, 27, 64, 125, 216, 343, …

על גבי איטרטור זה, עליכם ליצר איטרטור נוסיף המייצר את כל סכומי החזקות השלישיות בסדר אינסופי "אלכסוני":

0. 0^3+0^3
1. 1^3+0^3
2. 1^3+1^3
3. 2^3+0^3
4. 2^3+1^3
5. 2^3+2^3
6. 3^3+0^3
7. 3^3+1^3
8. 3^3+2^3
9. 3^3+3^3
10. 4^3+0^3
11. 4^3+1^3
12. 4^3+2^3
13. …

דוגמה ויזואלית ניתן למצוא כאן.

השתמשו באיטרטור שבניתם בשביל לחשב את מספר המונית. ~~התוכנית תדפיס שני מספרים טבעיים שונים, המציינים את הנקודה הראשונה שבה הפרמטר מחזיר שני ערכים זהים:~~

התכנית תחשב את מספר המונית, ותדפיס את המונים של הלולאות ושל התנאים, תוך שימוש בשיטה הבאה: מציאת שני המספרים הטבעיים השונים שבהם האיטרציה מחזירה ערכים זהים (אפשר להשתמש בספריה מתוך java.util אם צריך למציאת הזהויות).

Return i,j, i !=j, such that the parameter return the same number in iteration i, and in iteration j. Iterations are numbered 0, 1, ….

אפשר להתקין ולעבוד עם אקליפס, אבל התכנית הסופית צריכה להיות מורצת משורת הפקודה.

3. מספר המונית: תרגיל מסכם

במשימה זו, אתם תכתבו תכנית ב-PYTHON (גרסת 2.7, ולא אחרת) אשר מריצה את כל התכניות שכתבתם במהלך הקורס לחישוב מספר המונית (רשימת השפות מובאת בהמשך). על התוכנית להריץ כל גרסה 20 פעמים, ולייצר קובץ HTML שבו יש טבלה שיש לה העמודות הבאות:

1. שפת התכנות.
2. זמן הריצה הממוצע על פני ההרצות.
3. סטיית התקן המדגמית של זמני הריצה.
4. מספר ההשוואות (יתכן שיהיה חסר בשפת AWK).
5. מספר האיטרציות של לולאות.

אין לשנות את התכניות שהגשתם! יש להשתמש באותן תכניות בדיוק שהגשתם במהלך הקורס. אם מסיבה כלשהי אתם נאלצים לשנות את התכניות כדי להתאים אותן להרצה בשורות הפקודה, הוסיפו הערה לפני כל שינוי כזה בה יהיה כתוב:

LATE MODIFICATION: explain here the nature and reason of modification

הוסיפו לחלק היבש ניתוח של הממצאים: השוו את הערכים בטבלה, ואמרו משהו אינטליגנטי על התוצאות שהתקבלו. שפות עם טיפוסיות סטטית? דינמית? מהדר לעומת פרשן? שפה מסורבלת יותר או פחות? אלגוריתמים שונים?

3.1. הוראות הגשה לחלק זה

עליכם להגיש קובץ ZIP בשם cab.zip שבו יש תיקיות לכל אחת מהשפות:

1. AWK
2. Bash
3. ~~C~~
4. Java
5. ML
6. Nim
7. Pascal
8. ~~Prolog~~
9. Python

שימו לב: אין צורך לכתוב את תוכנית המונית בשפות C ופרולוג (לא ניתנו בתרגילים קודמים).

התיקיות חייבות להיות בדיוק בשמות אלו (**הקפידו על אותיות גדולות וקטנות**). תוכן התיקיות יהיה כרצונכם.

בתיקייה הראשית יהיו שני קבצים:

1. תכנית PYTHON בשם compile.py, שתפקידה לקמפל את כל מה שדורש קומפילציה.
2. תכנית PYTHON בשם compare.py, שמניחה שכל מה שדורש קומפילציה כבר קומפל, מריצה את כל התוכניות 20 פעמים כנדרש ובונה קובץ HTML (בתיקייה הראשית) שייקרא compare.html.

הניחו שהשורה הבאה תתבצע לפני הרצת התכניות: `chmod +x compare.py compile.py`.

3.2. הנחיות נוספות

בשביל להריץ את התוכניות שלכם באמצעות PYTHON, השתמשו בקוד המצורף לתרגיל. הפונקציה execute מקבלת פקודה, מריצה אותה ומחזירה לכם את הפלט של הריצה ואת זמן הריצה. בקובץ יש דוגמה שמבהירה את השימוש בפונקציה. אין צורך להגיש את הקובץ execute, מספיק להעתיק מתוכו את החלקים הרלוונטיים ולהשתמש בהם בקוד שלכם.

4. חלק מעשי - פרולוג

4.1. הנחיות כלליות

1. בכל הסעיפים ניתן להשתמש בכל הפרדיקטים שנלמדו בתרגולים, אך **רק בהם**. כל פרדיקט מובנה בשפת פרולוג שלא מופיע בתרגולים, תצטרכו ל**ממש בעצמכם**.
2. בכל סעיף מצוין במפורש אילו ארגומנטים הם קונקרטיים ואלו ארגומנטים יכולים להיות קונקרטיים או משתנים. שימו לב שאתם עונים בכל סעיף על הדרישות שלו.
3. אתם רשאים להוסיף פרדיקטי עזר כרצונכם בכל סעיפי התרגיל.
4. בכל סעיף, מותר (ולעתים אפילו רצוי) להשתמש בפרדיקטים הקודמים שבניתם בתרגיל.
5. בכל הסעיפים, אם המפרש מחזיר תשובה אחת נכונה או יותר, ואז מחכה לסימן ';' בשביל להמשיך את הריצה, ומיד לאחר מכן מחזיר false, הפתרון עדיין תקין.
6. לכל הטסטים יש Timeout של 60 שניות. הקלטים עליהם יבדקו הסעיפים בתרגיל הם סבירים, כלומר על כל מחשב אישי שנקנה בשבע השנים האחרונות או כל מחשב שנמצא בחווה צריך להריץ את הבדיקות בפחות מ-60 שניות. למעשה, 60 שניות הם מעין גבול עליון של זמן ריצה ועוד טווח ביטחון גדול, כאשר תבדקו את הקוד שלכם על קלטים סבירים, זמן הריצה צריך להיות פחות מ-5 שניות.

4.2. חימום

בחלק זה נייצג מספרים שלמים ואי-שליליים באמצעות "ייצוג-s" באופן הבא:

- המספר 0 ייוצג ע"י הספרה 0.
- המספר 1 ייוצג ע"י s(0).
- המספר 2 ייוצג ע"י s(s(0))
- וכן הלאה.

<u>ממשו את הפרדיקטים הבאים</u>:

1. כתבו פרדיקט בשם isS(X) המסתפק אם"ם X הוא מספר בייצוג שתואר.

   ניתן להניח ש-X הינו קונקרטי.

<u>דוגמאות הרצה</u>:

```
?- isS(0).
true.
```

```
?- isS(5).
false.

?- isS(s(0)).
true.

?- isS(s(s(a))).
false.
```

2. כתבו פרדיקט בשם s2int(X,Y) המסתפק אם X הוא ייצוג-s של המספר השלם והאי-שלילי Y.

ניתן להניח ש-X הינו קונקרטי, ו-Y יכול להיות קונקרטי או משתנה.

<u>דוגמאות הרצה</u>:

```
?- s2int(0, X).
X = 0.

?- s2int(s(0), X).
X = 1.

?- s2int(s(s(0)), X).
X = 2.
```

3. כתבו פרדיקט בשם add(X,Y,Z) המסתפק אם"ם Z הוא תוצאת החיבור של X ו-Y, כאשר X,Y,Z נתונים כולם בייצוג-s.

אם Z קונקרטי, אז X,Y יכולים קונקרטיים או משתנים (לאו דווקא ביחד. כלומר גם האפשרויות שבהן אחד מהם הוא קונקרטי והשני משתנה הן תקינות).

אם Z הוא משתנה, אז X,Y חייבים להיות קונקרטיים.

<u>דוגמאות הרצה</u>:

```
?- add(s(0), s(s(0)), Z).
Z = s(s(s(0))).

?- add(X, Y, s(s(0))).
X = 0,
Y = s(s(0)) ;
X = Y, Y = s(0) ;
X = s(s(0)),
Y = 0 ;
false.
```

שימו לב: במקרה שבו Z הוא קונקרטי, ו-X,Y הם משתנים, עליכם לוודא שכל הפתרונות חוזרים באמצעות שימוש ב-';' ושאין חזרה על פתרונות זהים, כפי שניתן לראות בדוגמת ההרצה (עם זאת, אין חשיבות לסדר של הופעת הפתרונות).

4.3. עץ פורש מינימום

4.3.1. סמי הכבאי

בשאלה זו נעזור לסמי הכבאי בבעיית תכנון חשובה. סמי התבקש על ידי עיריית חיפה לבחור מיקום לתחנת כיבוי אש עירונית. אנחנו נניח לצורך השאלה שנתון לנו גרף לא מכוון וקשיר $G=(V,E)$ שמייצג את נקודות העניין בעיר ואת הדרכים ביניהן, כך ש-V היא קבוצת הצמתים בגרף ו-E היא קבוצת הקשתות. בנוסף, על כל אחת מהקשתות מצוין משקל שלם ואי שלילי.

סמי החליט לפתור את הבעיה בשני שלבים:

1. בשלב הראשון, סמי מעוניין למצוא עץ פורש מינימום (עפ"מ) של הגרף (הסבר מיד).
2. בשלב השני, סמי מעוניין למצוא את הצמתים בעפ"מ שמצא בשלב הראשון, כך שאם הוא יציב בהן את תחנת הכיבוי, אז המרחק מתחנת הכיבוי לצומת הרחוק ביותר על פני העץ, תהיה הקצרה ביותר מבין כל הצמתים (דוגמה בהמשך).

4.3.2. עץ פורש מינימום

ראשית, נסביר מהו עץ פורש מינימום. נניח שנתון לנו גרף לא מכוון עם משקלים על הקשתות. אנחנו יודעים מלימודי הקומבינטוריקה שלנו שאם נתונה לנו קבוצה חסרת מעגלים של $|V|-1$ קשתות אז היא עץ. לחילופין, אם נתונה לנו קבוצה של $|V|-1$ קשתות כך שניתן להגיע מכל צומת בגרף לכל צומת אחר על פני קבוצת הקשתות (כלומר, תת

הגרף הינו קשיר), אז שוב מדובר בעץ. בהנחה שהגרף המקורי G הוא אכן קשיר, אז במקרה הגרוע עלול להיות מספר אקספוננציאלי של תתי קבוצות של קשתות שבהן יש בדיוק $|V|-1$ קשתות ומתקבל עבורן עץ. לכל קבוצת קשתות כזו אנחנו מחשבים את סכום המשקלים של הקשתות שבה. אמנם, קבוצת תתי הקבוצות של קשתות מהצורה הזו עלולה להיות גדולה מאוד, אבל היא סופית, ולכן בהכרח קיים עבורה מינימום. עכשיו אנחנו מגיעים להגדרה הפורמלית:

בהינתן גרף $G=(V,E)$ לא מכוון וקשיר, עץ פורש מינימום הוא גרף $G'=(V,E')$ כך ש-G' הינו עץ, ולא קיים $G''=(V,E'')$ כך שסכום המשקלים על פני קשתות E'' קטן ממש מסכום המשקלים על פני קשתות E'.

שימו לב שהאלגוריתם הנאיבי שעובר באופן סדרתי על כל תתי קבוצות הקשתות בגודל $|V|-1$, בודק האם הגרף הינו עץ והינו קשיר, ומחפש את עץ פורש מינימום עלול להיות אלגוריתם אקספוננציאלי. לשמחתנו, יש שני אלגוריתמים מפורסמים למדי למציאת עץ פורש מינימום שהם גם יעילים (רצים בזמן פולינומיאלי): הראשון נקרא Kruskal והשני נקרא Prim. בתרגיל הזה אנחנו נממש את אלגוריתם **Kruskal**.

4.3.3. אלגוריתם Kruskal:

i. אתחל קבוצה ריקה E'.
ii. מיין את הקשתות של הקבוצה E לפי המשקלים שעל הקשתות מקטנה לגדולה. נסמן את הסידור של הקשתות ב-O.
iii. לכל קשת e ב-O:
   a. אם הוספת הקשת e ל-E' **לא יוצרת** מעגל בגרף $G'=(V,E')$, הוסף את e ל-E'.

יחידות הפתרון: קל להשתכנע שאם משקלי הקשתות בגרף הם שונים, אז קיים עץ פורש מינימום יחיד. מי שמתעניין בהוכחה פורמלית יכול למצוא אותה כאן.

הנחה: לצורך פשטות, **אנחנו נניח בכל התרגיל שמשקלי הקשתות שונים זה מזה**, ולכן קיים עץ פורש מינימום יחיד בגרף.

כמעט סגרנו את הידע התיאורטי הנדרש לפתרון התרגיל, למעט דבר אחד: כיצד בודקים שהוספת קשת בשלב 3a לא יוצרת מעגל בגרף G'? יש שיטות שונות לבדוק זאת. אנחנו נממש אחת חביבה ויעילה שעושה שימוש במבנה הנתונים המגניב UnionFind. להזכירכם, מבנה הנתונים הזה דורש לממש שלוש פונקציות:

- הפונקציה init מקבלת רשימה של צמתים ומחזירה קבוצה של Singleton-ים (קבוצות שמכילות איבר אחד) לכל אחד מהצמתים ברשימה.
- הפונקציה find מקבלת צומת a, ומחזירה את הקבוצה שמכילה את a.
- הפונקציה union מקבלת שני צמתים a,b ומאחדת את הקבוצה שמכילה את a ואת הקבוצה שמכילה את b לקבוצה אחת.

בהינתן מבנה הנתונים הזה, קל לבדוק את התנאי ב-3a:

i. נסמן $e = (a, b)$ ואת מבנה הנתונים נסמן ב-U.

ii. התנאי לכך שהוספת הקשת e ל-E' **לא יוצרת** מעגל בגרף G' הוא (בפסאודו קוד):

```
find(U, a) != find(U, b)
```

אם התנאי הזה מתקיים (כלומר, אנחנו מוסיפים את הקשת e ל-E'), אז נאחד את הקבוצה שמכילה את a ואת הקבוצה שמכילה את b.

הדרך שלנו לייצג גרף בתרגיל היא באמצעות רשימה של יחסים מסוג:

```
edge(a, b, w)
```

כלומר, קיימת קשת מצומת a לצומת b ומשקלה הוא w.

אנחנו נניח ש**אין קשתות עצמיות**, כלומר לא קיימות קשתות מהצורה:

```
edge(a, a, w)
```

ואנחנו נניח גם ש**אין קשתות מקבילות או אנטי מקבילות**, כלומר אם קיימת קשת:

```
edge(a, b, w)
```

אז לא קיימת קשת נוספת מהצורה:

```
edge(a, b, w1)
```

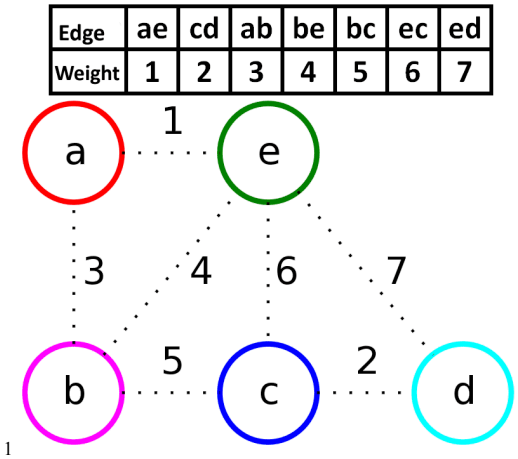
וגם לא קיימת קשת מהצורה:

```
edge(b, a, w2)
```

<u>שימו לב</u>: הגדרה זו לא מבטיחה יחידות של ייצוג של גרף. למשל, בהינתן רשימה כלשהי של קשתות, כל פרמוטציה של הרשימה, מתארת את אותו גרף בדיוק. כמו כן, אם ניקח קשת כלשהי edge(x, y, w) ונחליף אותה בקשת edge(y, x, w), נישאר עדיין עם אותו גרף. ודאו שנקודה זו ברורה לכם לפני שאתם ממשיכים.

4.3.4. דוגמה

נסתכל על הגרף הבא (בלינק <u>הזה</u> תוכלו למצוא גרסת gif שלו שמתארת ריצה של אלגוריתם Kruskal):



[1]

דרך אחת לייצג את הגרף היא באמצעות הרשימה הבאה:

```
[edge(a, b, 3), edge(a, e, 1), edge(b, c, 5), edge(b, e, 4),
edge(c, d, 2), edge(c, e, 6), edge(d, e, 7)]
```

לאחר הפעלת אלגוריתם Kruskal, אנחנו נקבל את העץ הפורש המינימלי הבא (ודאו שברור לכם למה לפני שאתם ממשיכים): :

```
[edge(a, e, 1), edge(c, d, 2), edge(a, b, 3), edge(b, c, 5)]
```

כעת, נותר לנו למצוא את המיקומים החוקיים של תחנת כיבוי האש.

קל לראות שאם נבחר לקבוע את השורש בצמתים d או e, נקבל שהמרחק מהשורש לצומת הרחוק ביותר בעץ הוא 11 .

שאר המרחקים המקסימליים מכל צומת:

- עבור הצומת a, נקבל 10.
- עבור הצומת c, נקבל 9.
- עבור הצומת b, נקבל 7.

לכן b הוא המיקום החוקי היחיד של תחנת כיבוי האש.

מספיק דיבורים, יאללה לעבודה.

---

[1] By Schulllz - Own work, CC BY-SA 3.0, https://commons.wikimedia.org/w/index.php?curid=30888975

4.3.5. כתיבת הקוד

1. ממשו את הפרדיקט הבא:

```
sortEdges(Edges, SortedEdges)
```

שמסתפק כאשר הרשימה SortedEdges היא רשימה ממוינת של הקשתות הנמצאות ברשימה Edges לפי משקלי הקשתות (מקטן לגדול).

ניתן להניח ש-Edges הינו קונקרטי, ו-SortedEdges יכול להיות קונקרטי או משתנה.

<u>הערה</u>: אתם חופשיים לבחור בכל שיטת מיון שעולה על דעתכם, ובלבד שזמן הריצה שלה הינו סביר. דוגמה לאלגוריתם עם זמן ריצה סביר הוא BubbleSort, ודוגמה לאלגוריתם עם זמן ריצה לא סביר הוא MonkeySort.

<u>דוגמת הרצה</u>:

```
?- sortedEdges([edge(a, b, 3), edge(a, e, 1), edge(b, c, 5), edge(b, e, 4), edge(c, d, 2), edge(c, e, 6), edge(d, e, 7)], SortedEdges).
SortedEdges = [edge(a, e, 1), edge(c, d, 2), edge(a, b, 3), edge(b, e, 4), edge(b, c, 5), edge(c, e, 6), edge(d, e, 7)].
```

2. ממשו את הפרדיקט הבא:

```
unique(L1, L2)
```

שמסתפק כאשר הרשימה L2 היא רשימה שמכילה את כל האיברים שנמצאים ב-L1 ללא חזרות.

ניתן להניח ש-L1 הינו קונקרטי, ו-L2 יכול להיות קונקרטי או משתנה.

<u>שימו לב</u>: סדר הופעת האיברים ב-L2 צריך להתאים לסדר האיברים ב-L1, במובן שאם ההופעה הראשונה של a ב-L1 קודמת להופעה הראשונה של b ב-L1, אז a יופיע לפני b ב-L2 (ראו דוגמאות הרצה).

<u>דוגמאות הרצה</u>:

```
?- unique([1,2,7,4,0,2,3,1,8,4,3], L).
L = [1, 2, 7, 4, 0, 3, 8].

?- unique([[], [1, 2], [2, 3], [], [1, 1], [1, 2], [1, 2, 3]], L).
L = [[], [1, 2], [2, 3], [1, 1], [1, 2, 3]].
```

3. ממשו את הפרדיקט הבא:

```
singletons(Edges, Singletons)
```

שמסתפק כאשר הרשימה Singletons היא רשימה של איברים מהצורה [a] כאשר a הוא צומת בגרף.

ניתן להניח ש-Edges הינו קונקרטי, ו-Singletons יכול להיות קונקרטי או משתנה.

<u>תזכורת</u>: אנחנו מניחים שהגרף הינו קשיר, ולכן לכל צומת a קיימת קשת e כך ש-$e = (a,b)$ או $e = (b,a)$, כאשר b הוא צומת אחר בגרף.

<u>הערה</u>: אין חשיבות לסדר של האיברים בתוך Singletons.

<u>דוגמת הרצה</u>:

```
?- singletons([edge(a, b, 3), edge(a, e, 1), edge(b, c, 5), edge(b, e, 4), edge(c, d, 2), edge(c, e, 6), edge(d, e, 7)], Singletons).
Singletons = [[a], [b], [e], [c], [d]].
```

4. ממשו את הפרדיקט הבא:

```
find(A, UnionFind, SetA)
```

שמסתפק כאשר הרשימה SetA היא הקבוצה מתוך UnionFind שמכילה את A.

ניתן להניח ש-A, UnionFind הם קונקרטיים, ו-SetA יכול להיות קונקרטי או משתנה.

הנחה: הניחו ש-UnionFind הוא רשימה של רשימות זרות, כלומר לא ייתכן שצומת יופיע בשתי קבוצות שונות.

שימו לב: אין חשיבות לסדר הופעת הצמתים ברשימה שחוזרת.

דוגמאות הרצה:

```
?- find(a, [[a, b], [e], [c, d]], S).
S = [a, b].

?- find(e, [[a, b], [e], [c, d]], S).
S = [e].
```

5. ממשו את הפרדיקט הבא:

```
union(A, B, UnionFind, UnionFind1)
```

שמסתפק כאשר הרשימה UnionFind1 מתקבלת על ידי איחוד של הקבוצות שמכילות את A ואת B מתוך UnionFind, ובלבד ש-A ו-B לא נמצאים באותה הקבוצה.

ניתן להניח ש-A, B, UnionFind הינם קונקרטיים, ו-UnionFind1 יכול להיות קונקרטי או משתנה.

שימו לב: אין חשיבות לסדר הופעת הרשימות בתוך הרשימה החיצונית, ואין חשיבות לסדר הופעת הצמתים בכל אחת מהרשימות הפנימיות.

דוגמאות הרצה:

```
?- union(a, e, [[a, b], [e], [c, d]], U).
U = [[a, b, e], [c, d]].

?- union(c, d, [[a, b], [e], [c, d]], U).
false.
```

6. ממשו את הפרדיקט הבא:

```
minimumSpanningTree(Edges, MST)
```

שמסתפק כאשר הרשימה MST מכילה קשתות של עץ פורש מינימום של הגרף הנתון על ידי Edges.

ניתן להניח ש-Edges הינו קונקרטי, ו-MST הינו משתנה.

<u>תזכורת</u>: אנחנו מניחים שמשקלי הקשתות שונים ובתוספת לכך שהגרף קשיר, אז קיים עץ פורש מינימום יחיד, ולכן הפרדיקט מחזיר את **ה**עץ **ה**פורש **ה**מינימום של הגרף.

<u>שימו לב</u>: אין חשיבות לסדר הופעת הקשתות ב-MST.

<u>דוגמת הרצה</u>:

```
?- minimumSpanningTree([edge(a, b, 3), edge(a, e, 1),
edge(b, c, 5), edge(b, e, 4), edge(c, d, 2), edge(c, e, 6),
edge(d, e, 7)], MST).
MST = [edge(a, e, 1), edge(c, d, 2), edge(a, b, 3), edge(b,
c, 5)].
```

7. ממשו את הפרדיקט הבא:

```
directedTree(A, MST, DirectedTree)
```

שמסתפק כאשר הרשימה DirectedTree מכילה את הקשתות של MST כך שהקשתות שייכות לעץ מכוון שהשורש שלו הוא הצומת A (ראו דוגמה).

ניתן להניח ש-A, MST הינם קונקרטיים, ו-DirectedTree הינו משתנה.

<u>דוגמאות הרצה</u>:

```
?- directedTree(a, [edge(a, e, 1), edge(c, d, 2), edge(a,
b, 3), edge(b, c, 5)], DirectedTree).
```

```
DirectedTree = [edge(a, e, 1), edge(a, b, 3), edge(b, c, 5), edge(c, d, 2)].
```

```
?- directedTree(b, [edge(a, e, 1), edge(c, d, 2), edge(a, b, 3), edge(b, c, 5)], DirectedTree).
DirectedTree = [edge(b, a, 3), edge(b, c, 5), edge(c, d, 2), edge(a, e, 1)].
```

שימו לב: אין חשיבות לסדר הופעת הקשתות ב-DirectedTree.

שימו לב: באופן כללי, בהינתן עץ לא מכוון, בחירה של שורש משרה עץ מכוון יחיד. מטרת הפרדיקט היא לסדר את כיוון הקשתות כך שתהיה חשיבות לסדר שלהן בהתאם לעץ המכוון המתקבל.

8. ממשו את הפרדיקט הבא:

```
longestPath(A, MST, M)
```

שמסתפק כאשר M הוא המקסימום על פני סכום המשקולות על פני המסלול מצומת A לכל צומת אחר בעץ לפי העץ המכוון המושרה על ידי A על פני MST.

ניתן להניח ש-A, MST הינם קונקרטיים, ו-M יכול להיות קונקרטי או משתנה.

לא הנחה: לא ניתן להניח ש-MST הוא עץ מכוון חוקי. השתמשו בסעיף 7 בשביל להפוך את העץ למכוון לפי צומת A.

תזכורת: בעץ יש מסלול יחיד מכל צומת לכל צומת.

דוגמאות הרצה:

```
?- longestPath(a, [edge(a, e, 1), edge(a, b, 3), edge(b, c, 5), edge(c, d, 2)], M).
M = 10 .
```

```
?- longestPath(e, [edge(a, e, 1), edge(a, b, 3), edge(b, c,
5), edge(c, d, 2)], M).
M = 11 .
```

9. ממשו את הפרדיקט הבא:

```
firestations(Edges, Vertices)
```

שמסתפק כאשר Vertices היא רשימה של כל הצמתים שאם סמי ימקם בהם את תחנת הכיבוי, אז המרחק המקסימלי מכל צומת כזה לשאר הצמתים בעץ פורש מינימלי של הגרף Edges יהיה הקטן ביותר.

ניתן להניח ש-Edges הינו קונקרטי, ו-Vertices הינו משתנה.

<u>שימו לב</u>: אין חשיבות לסדר הופעת הצמתים ב-Vertices.

<u>דוגמת הרצה</u>:

```
?- firestations([edge(a, b, 3), edge(a, e, 1), edge(b, c,
5), edge(b, e, 4), edge(c, d, 2), edge(c, e, 6), edge(d, e,
7)], Vertices).
Vertices = [b].
```

5. הנחיות הגשה

- בתרגיל זה ניתן להשתמש רק בחומר שנלמד עד תרגול 13 (כולל). אין להשתמש באף פונקציה/תכונה של השפה שלא נלמדה בתרגולים.
- קובץ הקוד הנדרש בתרגיל זה הינו ex6.pl.
- קובץ ה-zip שתגישו יכיל את הקבצים הבאים: dry.pdf, ex6.pl, cab.zip.
- ככל שיהיו תיקונים או הבהרות לתרגיל זה, הם יופיעו בצבע כתום על גבי התרגיל, וכמו כן השינויים ירוכזו בדף הבא.

הוראות הגשה כלליות ניתן למצוא כאן.

בהצלחה!

תיקונים והבהרות

04/06/16: תוקנה ההנחיה לגבי הפרמטרים של הפרדיקט union (חלק מעשי, סעיף 5). .

06/06/16: בוטלה תוכנית המונית בשפות C ופרולוג בתרגיל המסכם של מספר המונית.

07/06/16: עץ פורש מינימלי, סעיף 8, הדוגמה וההסבר לא היו תואמים. הדוגמה נכונה: העץ יכול להיות כלשהו, ולכן צריך לסדר אותו קודם כך שיהיה עץ מכוון ואז הדוגמאות מסתדרות.