

תוכנה 1 – חורף 2021/22

תרגיל מספר 7

הנחיות כלליות:

קראו בעיון את קובץ נהלי הגשת התרגילים אשר נמצא באתר הקורס.

את התרגיל הבא צריך להגיש באופן הבא:

- הגשה במערכת ה-Git תבצע על פי ההנחיות שראיתם בתרגול 0. צרו את ה repository שלכם מתוך הקישור הבא:

<https://classroom.github.com/a/XhSeDyFL>

יש לוודא שבתיקיית הגיט שלכם נמצאים הקבצים הבאים:

- א. קובץ פרטים אישיים בשם details.txt המכיל את שם המשתמש שלכם ב Moodle ואת מספר תעודת הזהות שלכם.
- ב. הקבצים Greedy.java, Graph.java, FractionalKnapSack.java. ניתן גם להשאיר את הקבצים Coins.java ו-GreedyAlgorithmTester.java, אך לא חובה, כיוון שקבצים אלו אינם נבדקים (הם נועדו לסייע לכם בבדיקה עצמית).
- ג. אין לצרף קבצים ותיקיות נוספות.

- הגשה במערכת ה Moodle (<http://moodle.tau.ac.il>): עליכם להגיש את קובץ הטקסט assignment.txt ובו קישור ל git repository שלכם.
- נא לא להשתמש בפקודה System.exit()! היא מחבלת בבדיקות אוטומטיות. אין כל צורך לעשות בה שימוש, כאשר תוכניות יכולות להסתיים ע"י הגעה לסוף מתודת ה-main.

בתרגיל זה נבנה פלטפורמה גנרית לפתרון בעיות אלגוריתמיות ע"פ השיטה החמדנית באמצעות שימוש במנשקים. בנוסף, במהלך התרגיל נשתמש באיטרטורים, בפונקציות מיון, ובמחלקות גנריות.

הערות:

1. מותר להוסיף משתנים חדשים ופונקציות חדשות למחלקות. על המשתנים והפונקציות להיות בעלות נראות פרטית.
2. מותר להשתמש ב Collections.sort().
3. מותר (ובמקרים מסוימים נחוץ) לכתוב מתודות עזר על מנת למנוע שכפול קוד, אך יש לשמור את כולן באותו הקובץ, ולא ליצור עבורן מחלקות חדשות.
4. אסור למחוק או לערוך מתודות ומשתנים שקיימים בשלד, אלא אם נכתב במפורש שמותר.
5. כל הפונקציות שתכתבו בתרגיל זה צריכות לרוץ בזמן פולינומיאלי בקלט.

אלגוריתם חמדן (שאלה 1)

אלגוריתם הוא דרך שיטתית וחד-משמעית לביצוע של משימה מסוימת, במספר סופי של צעדים. ניתן לחלק את סוגי האלגוריתמים למשפחות – ע"פ דרך הפעולה שלהן. למשל, משפחת האלגוריתמים הרקורסיביים מכילה בתוכה כל אלגוריתמים שניתן לממש בצורה רקורסיבית. דוגמא נוספת למשפחת אלגוריתמים היא משפחת האלגוריתמים החמדניים.

אלגוריתם חמדן, הוא כזה שבכל צעד עושה את הבחירה הטובה ביותר האפשרית, ולא מתחרט בהמשך. גישה זו נראית פשטנית מדי, וכמובן שלא תמיד היא נכונה, אך במקרים רבים היא מוצאת פתרון אופטימלי. בתרגיל זה נעסוק בפתרון בעיות בהן גישה החמדנית מחזירה פתרון אופטימלי.

בצורה כללית, האלגוריתם החמדן מורכב מארבע פונקציות שונות:

1. Selection function: פונקציה שמחזירה את האיבר הטוב ביותר (ע"פ העיקרון החמדן) אותו יש להוסיף.
2. Feasibility function: פונקציה שבודקת האם תוספת האיבר נותנת תוצאה חוקית.
3. Assign function: פונקציה שמוסיפה איבר לפתרון.
4. Solution function: פונקציה שבודקת האם פתרון הוא תקין.

פסאודו קוד כללי לאלגוריתם חמדן:

```
greedy_algorithm:
----candidates_list = ∅

----while iterator in selection() has more elements:

-----if feasibility(candidates_list, element) is True:
        assign(candidates_list, element):

-----if solution(candidates_list) is True:
        return candidates_list

----return null
```

עליכם לממש את הפונקציה `greedyAlgorithm` בממשק `Greedy` כך שכל מחלקה שמרחיבה את הממשק תוכל להפעיל את האלגוריתם החמדן.

לצורך בדיקה עצמית של הפונקציה, תוכלו להיעזר בקוד `Coins.java` שמייצג את בעיית ספירת המטבעות המינימלית. הסבר על הבעיה נמצא בעמוד הבא.

ספירת מטבעות מינימלית

בבעיית ספירות המטבעות יש לנו מלאי בלתי מוגבל של 5 סוגי מטבעות – בשווי 1, 2, 5, ו-10. כקלט לבעיה, אנו נקבל סכום יעד.

פתרון לבעיה הוא כמות המטבעות שצריך מכל סוג כך שסכום המטבעות שווה לסכום היעד. לדוגמא, עבור הקלט 15, פתרון אפשרי לבעיה הוא $[0,0,3,0]$. כלומר, 0 מטבעות בשווי 1, 0 מטבעות בשווי 2, 3 מטבעות בשווי 5, ו-0 מטבעות בשווי 10. פתרונות אפשריים נוספים לבעיה הם $[0,0,1,1]$, $[1,7,0,0]$, ו- $[15,0,0,0]$ (ישנם פתרונות נוספים).

פתרון אופטימלי לבעיה יהיה פתרון בעל מספר מינימלי של מטבעות. כלומר, פתרון בו סכום האיברים במערך הוא מינימלי. בדוגמא בה הקלט הוא 15, פתרון אופטימלי הוא $[0,0,1,1]$ (ובפרט, בבעיה זו, זהו הפתרון האופטימלי היחיד).

האלגוריתם החמדן עבור בעיה זו ינסה להוסיף, כל עוד לא עבר את סכום היעד, כמה שיותר מטבעות מהשווי הגבוה ביותר. ניתן להוכיח (לא בקורס שלנו) שעבור בעיית ספירת המטבעות המינימלית, האלגוריתם החמדן נותן פתרון אופטימלי.

מצורף לתרגיל הקוד Coins.java בו מימוש המנשק Greedy עבור בעיה זו. מומלץ לעיין בפתרון כדי להבין טוב יותר את דרישות התרגיל.

בעיית התרמיל השברי (שאלה 2)

הקדמה:

בבעיית התרמיל יש לנו מלאי של n חפצים. לכל חפץ יש משקל (weight) ולכל חפץ ערך (value). בנוסף, יש לנו תרמיל ובו מקום מוגבל לחפצים (capacity). נרצה להכניס כמות חפצים שלא תעבור את מגבלת התרמיל, כך ששווי כל החפצים הוא מקסימלי. בגרסה השברית של הבעיה (הגרסה בה אנחנו עוסקים), המשקל של כל חפץ (weight) הוא מספר עשרוני, וניתן לקחת חלק ממנו.

לדוגמא, עבור תרמיל שיכול לשאת עד 50 ק"ג ועבור החפצים $\{60, 10\}$, $\{100, 20\}$, $\{120, 30\}$ (כאשר האיבר הימני הוא המשקל והאיבר השמאלי הוא הערך) שלושת החלוקות הבאות הן פתרונות:

1. 20 ק"ג מחפץ בשווי 100, 30 ק"ג מחפץ בשווי 120. סה"כ שווי תכולת התרמיל הוא $100 + 120 = 220$.
2. 10 ק"ג מחפץ בשווי 60, 10 ק"ג מחפץ בשווי 100, 30 ק"ג מחפץ בשווי 120. סה"כ שווי תכולת התרמיל הוא $230 = \frac{10}{10} \cdot 60 + \frac{10}{20} \cdot 100 + \frac{30}{30} \cdot 120$.
3. 10 ק"ג מחפץ בשווי 60, 20 ק"ג מחפץ בשווי 100, 20 ק"ג מחפץ בשווי 120. סה"כ שווי תכולת התרמיל הוא $240 = \frac{10}{10} \cdot 60 + \frac{20}{20} \cdot 100 + \frac{20}{30} \cdot 120$.

נשים לב שמבין שלושת הפתרונות שהוצגו, הפתרון השלישי הוא בעל הערך הגבוה ביותר. פתרון אופטימלי לבעיה יהיה פתרון בעל ערך מקסימלי מבין כל הפתרונות האפשריים.

המחלקה FractionalKnapSack:

המחלקה FractionalKnapSack מייצגת את התרמיל, ואת n החפצים. ייצוג התרמיל נעשה ע"י מספר טבעי `int` שמגדיר את מגבלת המקום בתרמיל. ייצוג חפץ נעשה ע"י המחלקה הפנימית FractionalKnapSack.Item שמקבלת בבנאי המחלקה משקל (weight), ושווי (value).

האלגוריתם החמדן:

נמייין את החפצים לפי המשקל היחסי שלהם (לפי $value/weight$). לאחר מכן נעבור על החפצים (ע"פ הסדר שמיינו אותם) ונוסיף כל חפץ בשלמותו, כל עוד הוא נכנס בתוך התרמיל. כאשר הגענו לחפץ שלא נכנס למקום שנשאר בתרמיל, ניקח ממנו את החלק היחסי שנכנס לתרמיל ונעדכן את הערך של החלק שלקחנו. ניתן להוכיח (לא בקורס שלנו) שעבור בעיית התרמיל השברי, האלגוריתם החמדן נותן פתרון אופטימלי.

מאחר שהאלגוריתם חמדן, נוכל להתאים אותו למבנה הכללי שראינו:

1. Selection function: פונקציה שמחזירה את האיבר הטוב ביותר (ע"פ העיקרון החמדן) אותו יש להוסיף. כלומר, תחזיר איטרטור עבור החפצים כך שהמשקל היחסי מתחיל מהגבוה ביותר, עד הנמוך ביותר (ניתן להניח שהמשקל היחסי של כל החפצים שונים).
2. Feasibility function: פונקציה שבודקת האם תוספת האיבר נותנת תוצאה חוקית. כלומר, תבדוק האם יש עדיין מקום בתרמיל.
3. Assign function: פונקציה שמוסיפה איבר לפתרון. כלומר, מוסיפה חפץ לתרמיל.
4. Solution function: פונקציה שבודקת האם פתרון הוא תקין. כלומר, בודקת אם הגענו לסף קיבולת התרמיל או שלא נותרו חפצים למלא את התרמיל.

עליכם לממש את הממשק Greedy במחלקה FractionalKnapSack כך שבהרצת הפונקציה `greedyAlgorithm` על אובייקט תרמיל שברי נקבל תת-קבוצה של חפצים שמהווים פתרון מקסימלי.

על הפלט של `greedyAlgorithm` להיות בסדר שהוכתב ע"י ה `selection function`.

מציאת עץ פורש מינימלי (שאלה 3)

הקדמה:

בתורת הגרפים, גרף הוא ייצוג מופשט של קבוצה של אובייקטים, כאשר כל זוג אובייקטים בקבוצה עשויים להיות מקושרים זה לזה.

האובייקטים הניתנים לקישור מכונים צמתים או קודקודים. קבוצת הקודקודים מסומנת באות V . הקישורים בין הקודקודים מכונים צלעות או קשתות. קבוצת הצלעות מסומנת באות E . קבוצת הצלעות מקיימת $E \subseteq V \times V$.

בנוסף, בשאלה זו נעסוק בגרף בעל קשתות לא מכוונות. לכן, הקשת (u, v) והקשת (v, u) זהות.

גרף מוגדר ע"י קבוצת צמתים וקבוצת צלעות $G = (V, E)$. בשאלה זו נתמקד בגרף ממושקל בו לכל צלע $e \in E$ יש משקל אותו נסמן ב- $w(e)$.

נאמר שגרף הוא עץ אם הוא קשיב (קיים מסלול שמקשר כל זוג צמתים) וללא מעגלים (לא קיים מסלול שיוצא מצומת v וחוזר אליה).

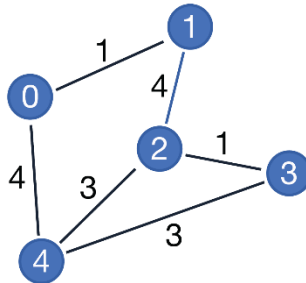
נאמר שתת-קבוצה של קשתות פורשת אם לכל $v \in V$ קיים $e \in E$ כך ש- v נמצא ב- e .

עץ פורש מינימלי:

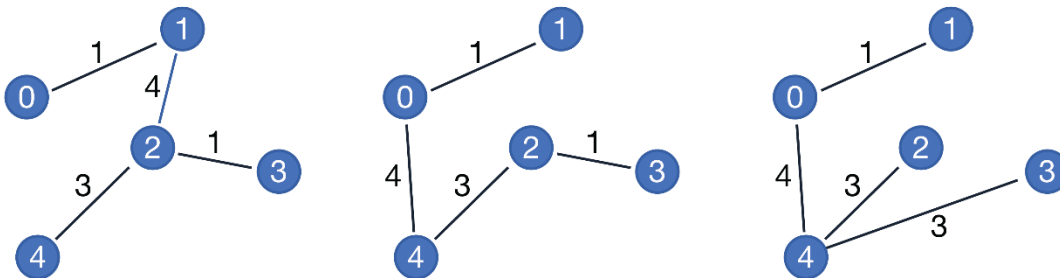
בבעיית העץ הפורש המינימלי אנו צריכים למצוא תת-קבוצה של קשתות כך שמקיימות:

1. תת-הקבוצה היא עץ.
2. תת-הקבוצה היא פורשת.
3. סכום המשקלים של הקשתות בתת-הקבוצה היא מינימלית.

לדוגמא, עבור הגרף:

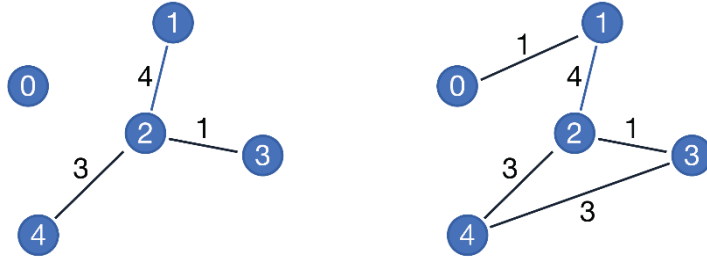


שלושת הגרפים הבאים הם עצים פורשים:



ושני העצים השמאליים הם בעלי משקל מינימלי (קיימים עצים נוספים בעלי משקל מינימלי).

לעומת זאת, שני הגרפים הבאים אינם עצים פורשים:



הגרף הימני אינו עץ מאחר שיש בו מעגל, והשמאלי לא פורש את הגרף המקורי.

המחלקה Graph:

המחלקה Graph מייצגת גרף. ייצוג קודקוד בגרף נעשה ע"י מספר טבעי `int`. ייצוג קשת (לא מכוונת) ממושקלת בגרף נעשית ע"י המחלקה הפנימית `Graph.Edge` שמקבלת בבנאי המחלקה שני קודקודים `(node1, node2)` שמייצגים קשת, ואת משקל הקשת (`weight`). בעת יצירת אובייקט `Graph`, כקלט לבנאי, נקבל מספר `n` שמגדיר את קבוצת הצמתים $V = \{0, \dots, n\}$, ונקבל רשימה של קשתות עם משקל.

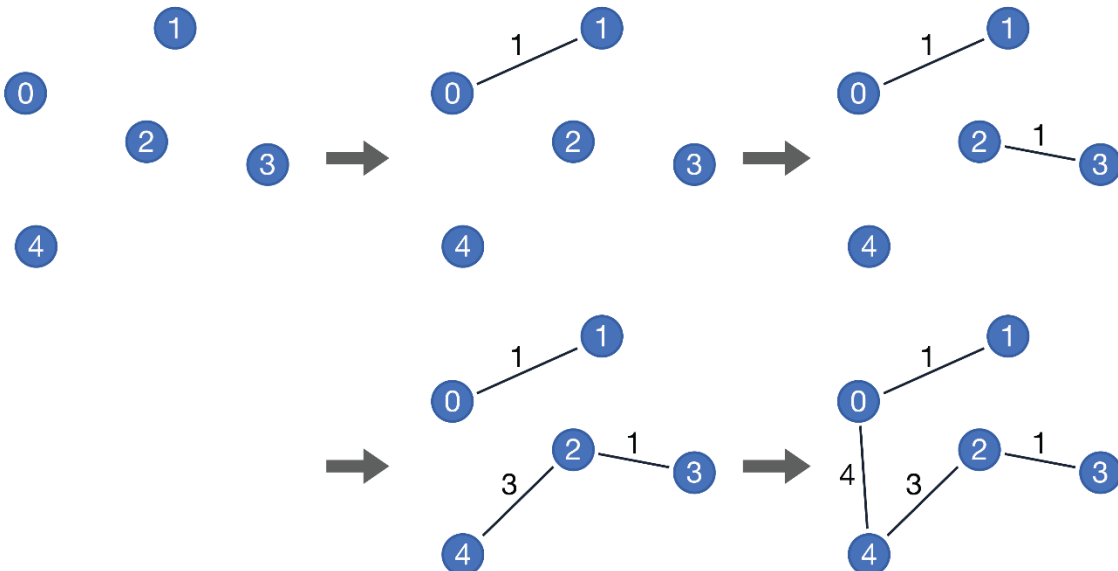
האלגוריתם של קרוסקל:

האלגוריתם של קרוסקל הוא אלגוריתם חמדן לפתרון בעיית מציאת עץ פורש מינימלי בגרף ממושקל קשיר לא מכוון.

האלגוריתם מבצע:

1. אתחול מערך קשתות ריק – `candidates`.
 2. מיון הקשתות כך שהמשקל, כך שהקשתות הקלות ראשונות.
 3. מעבר על הקשתות (מהקלה לכבדה): אם הקשת שנבחרה לא סוגרת מעגל, נוסיף אותה ל `candidates`.
 4. נחזור על השלב הקודם עד שיש במערך `candidates` כמות של $|V| - 1$ קשתות.
- האלגוריתם הוא חמדן, כיוון שבכל צעד נבחרת הפעולה הנראית אופטימלית באותו שלב - נוטלים את הקשת שמשקלה הקטן ביותר. ניתן להוכיח (לא בקורס שלנו) שעבור בעיית העץ הפורש המינימלי, האלגוריתם החמדן נותן פתרון אופטימלי.

את שלבי הפעולה של האלגוריתם אפשר לראות בציור הנ"ל:



- מאחר שהאלגוריתם של קרוסקל הוא חמדן, נוכל להתאים אותו למבנה הכללי שראינו:
1. Selection function: פונקציה שמחזירה את האיבר הטוב ביותר (ע"פ העיקרון החמדן) אותו יש להוסיף. כלומר, תחזיר איטרטור עבור הקשתות בגרף כך שמשקל הקשתות מתחיל מהנמוך ביותר, עד הגבוה ביותר.
 2. בנוסף, עבור שתי קשתות בעלות אותו המשקל – שובר השוויון יהיה לפי הצומת הראשון בקשת ואח"כ לפני הצומת השני. כלומר, עבור שתי קשתות בעלות בשקל זהה, הקשת (1,4) תהיה לפני (2,3), והקשת (1,3) תהיה לפני (1,4).
 3. Feasibility function: פונקציה שבודקת האם תוספת האיבר נותנת תוצאה חוקית. כלומר, תבדוק האם הוספת קשת חדשה יוצרת מעגל.
 4. Assign function: פונקציה שמוסיפה איבר לפתרון. כלומר, מוסיפה קשת לרשימת הקשתות שמהוות עץ פורש מינימלי.
 4. Solution function: פונקציה שבודקת האם פתרון הוא תקין. כלומר, הפונקציה בודקת האם קבוצת קשתות מהווה עץ פורש.

עליכם לממש את הממשק Greedy במחלקה Graph כך שבהרצת הפונקציה greedyAlgorithm על אובייקט גרף נקבל תת-קבוצה של קשתות שמהוות עץ פורש מינימלי.
 על הפלט של greedyAlgorithm להיות בסדר שהוכתב ע"י ה selection function.

טסטר

לתרגיל זה מצורפת מחלקת טסטר בו תסריטי בדיקות בסיסיים לכל אחת מהמחלקות. מומלץ להוסיף בדיקות משלכם.

בהצלחה!