הנחיות לפתרון תרגילי הבית

- על הקוד המוגש להיות מתועד היטב ועליו לכלול:
  - מפרט, כפי שהודגם בתרגול.
  - תיעוד של כל מחלקה ומתודה ושל קטעי קוד רלוונטיים.
  - במידת הצורך, יש להוסיף תיעוד חיצוני.
- יש להפעיל את הכלי Javadoc כדי ליצור קבצי תיעוד בפורמט HTML ולצרף אותם לפתרון הממוחשב המוגש. כדי לגרום לקובצי ה-HTML להכיל את פסקאות המפרט שבהן אנו משתמשים, יש לציין זאת במפורש. ב-Eclipse, ניתן לבצע פעולה זו באופן הבא: 1. לבחור Export מתפריט File, לבחור Java->Javadoc וללחוץ על כפתור Next, 2. לבחור עבור Javadoc command את הקובץ javadoc.exe מתוך התיקייה bin הנמצאת בתיקייה שבה מותקן ה-Java SDK, 3. לבחור את הקבצים שלהם מעוניינים ליצור תיעוד וללחוץ פעמיים על כפתור Next, 4. להקיש ב-Extra Javadoc options את השורה הבאה וללחוץ על כפתור Finish:

-tag requires:a:"Requires:" -tag modifies:a:"Modifies:" -tag effects:a:"Effects:"

- התנהגות ברירת המחדל של פעולות assert היא disabled (הבדיקות לא מתבצעות). כדי לאפשר את הידור וביצוע פעולות assert, יש לבצע ב-Eclipse את הפעולות הבאות: 1. מתפריט Run לבחור Debug Configurations , 2. בחלון שנפתח, לעבור ללשונית Arguments, 3. בתיבת הטקסט VM arguments לכתוב ea–, 4. ללחוץ על כפתור Debug.

הנחיות להגשת תרגילי בית

- תרגילי הבית הם חובה.
- ההגשה בזוגות בלבד.
- עם סיום פתירת התרגיל, יש ליצור קובץ דחוס להגשה המכיל את:
  - כל קבצי הקוד והתיעוד.
  - פתרון לשאלות ה״יבשות״ בקובץ Word או PDF. על הקובץ להכיל את שמות ומספרי תעודות הזהות **של שני הסטודנטים** המגישים.
- הגשת התרגיל היא **אלקטרונית בלבד**, דרך אתר הקורס **ע״י אחד מבני הזוג בלבד**.

- תרגיל שיוגש באיחור וללא אישור מתאים (כגון, אישור מילואים), יורד ממנו ציון באופן אוטומטי לפי חישוב של 2 נקודות לכל יום איחור.
- על התוכנית לעבור קומפילציה. על תכנית שלא עוברת קומפילציה יורדו 30 נקודות.

מועד ההגשה:
יום ד', 30.12

המטרות של תרגיל בית זה הן:

- להתנסות בתכן ADT (Abstract Data Type), בכתיבת מפרט עבורו, במימושו ובכתיבת קוד המשתמש בו.
- לתרגל ביצוע בדיקות יחידה בעזרת JUnit.

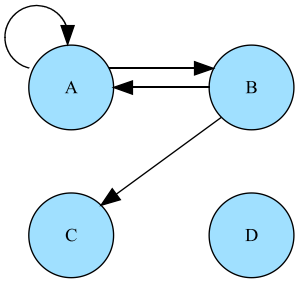
הצגת הבעיה

בתרגיל בית זה תיצרו הפשטה שתאפשר למצוא מסלול קצר ביותר בין שתי נקודות. לשם כך תזדקקו להפשטה עבור גרף, המייצג את הקישוריות בין נקודות, להפשטה עבור מסלול, המייצג את מחיר המעבר דרך אוסף נקודות בגרף בסדר מסוים, ולאלגוריתם למציאת מסלול בעל המחיר הנמוך ביותר האפשרי.

את ההפשטה עבור גרף ועבור מציאת מסלול קצר ביותר יהיה עליכם לתכנן בעצמכם. ההפשטה עבור מסלול נתונה לכם בממשק Path. ה-test driver המסופק, שאת מימושו תצטרכו להשלים, יבצע בדיקות, שכל אחת מהן תורכב מרשימת פקודות. פקודות אלה יתקבלו מאמצעי קלט (המקלדת/קובץ) ותוצאותיהן תשלחנה לאמצעי פלט (המסך/קובץ).

שאלה 1 (55 נקודות)

גרף מכוון (directed graph) מורכב מאוסף של *צמתים (nodes)*, שחלקם עשויים להיות מקושרים בעזרת *קשתות (edges)*. לכל קשת יש כיוון, כלומר, עשוי להתקיים מצב, כמו בדוגמה הבאה, בו קיימת קשת המקשרת בין הצומת B לצומת C, אך לא קיימת קשת המקשרת בין הצומת C לצומת B.



בתרגיל זה נניח כי לא יכולה להיות יותר מקשת אחת המקשרת צומת מסוים לצומת אחר (אך יכולה, כמובן, להיות קשת בכיוון ההפוך).

הבנים (children) של B הם הצמתים שאליהם יש קשת מ-B. בדוגמה הנ"ל, הבנים של B הם A ו-C. *האבות (parents)* של B הם הצמתים שיש קשת מהם ל-B. בדוגמה הנ"ל, האב היחיד של B הוא A.

בשאלה זו תעסקו ביצירת טיפוס נתונים מופשט עבור גרף מכוון בעל משקלות (מחירים) לצמתים ובבדיקתו.

א.
עליכם להתחיל בכך שתחליטו על הפעולות שעל הפשטה זו לכלול. לשם כך, ניתן להיעזר באוסף הפקודות האפשרי בקובצי הבדיקה, המוגדר בנספח בסוף התרגיל. כתבו מפרט עבור ההפשטה שבחרתם, כולל פסקאות @requires, @modifies ו-@effects. את המפרט יש לרשום בקובץ בשם Graph.java.

הנחיות:
1. אובייקט שהוא מופע של Graph צריך לאפשר אכסון צמתים מטיפוס כלשהו. בפרט, צמתים אלה יכולים להיות טיפוסים מורכבים שמכילים פונקציונליות נוספת (למשל, מחשבים ברשת תקשורת).
2. מופע של Graph לא צריך לטפל ולא להיות מודע למחירי הצמתים. מחירים אלו יאוכסנו בתוך הצמתים עצמם.

להגשה ממוחשבת: המפרט של המחלקה Graph.
להגשה "יבשה": תיעוד חיצוני המסביר את השיקולים בבחירת הפעולות השונות של ההפשטה עבור גרף. הסבירו מדוע אוסף פעולות זה נראה לכם מספק לפתרון הבעיה הנתונה.

ב.
ממשו את המפרט שיצרתם בסעיף א'. בעת המימוש יש לזכור כי אלגוריתם למציאת מסלול קצר ביותר יעשה שימוש במימוש זה. אנו מעוניינים במימוש של Graph שהאלגוריתם למציאת מסלול קצר ביותר יתבצע עליו בסיבוכיות חישוב סבירה. אלגוריתם למציאת מסלול קצר ביותר משתמש באופן תכוף בפעולה של מציאת רשימת הבנים של צומת מסוים. עליכם לרשום מימוש שיבצע פעולה זו בזמן קבוע. גם הפעולות לבניית גרף צריכות להתבצע בזמן סביר. עם זאת, ראשית עליכם לדאוג לתכן נכון ורק לאחר מכן לביצועים טובים.

יש לרשום abstraction function ו-representation invariant בתוך שורות הערה בקוד של Graph. בנוסף, יש לממש מתודת checkRep() לבדיקת ה-representation invariant ולקרוא לה במקומות המתאימים בקוד.

הנחיות:
1. ב-Java Documentation ניתן למצוא את פירוט סיבוכיות החישוב של פעולות שונות של כל אחד מהמכלים הקיימים בשפת Java. ניתן להשתמש בנתונים אלה לשם הערכת סיבוכיות החישוב של מימושים שונים.
2. למרות שהפלט עבור הבדיקות של גרף מוגדר לעתים כרשימה של צמתים לפי סדר אלפביתי, אין זה אומר שהמימוש צריך להחזיר או להכיל צמתים לפי סדר זה. ניתן,

לחילופין, למיין את רשימת הצמתים לפני הצגתה. לשם כך ניתן להשתמש במתודה sort() של המחלקה java.util.Collections.

להגשה ממוחשבת: מימוש המחלקה Graph.
להגשה "יבשה": תיעוד חיצוני המסביר את השיקולים שהובילו למימוש הנבחר. הציעו **במילים** מימוש נוסף והשוו אותו לזה שבחרתם.

ג.
בדיקת המימוש של Graph תתבצע בעזרת test driver, ששלד מימוש שלו נתון במחלקה TestDriver. מחלקה זו קוראת אוסף פקודות המתקבלות ממקור קלט (למשל, קובץ או המקלדת), מפעילה אותן, ושולחת את התוצאות לאמצעי פלט (למשל, קובץ או המסך). עליכם להשלים את שלד מימוש זה במקומות המסומנים. יש להשתמש בשדה output לצורך ביצוע הפלט. לצורך בדיקת המימוש יש להשתמש במופעים של המחלקה הנתונה WeightedNode עבור צמתים בגרף. המבנה התחבירי של אוסף הפקודות החוקיות ותוצאותיהן מוגדר בנספח בסוף התרגיל.

נתונה המחלקה ScriptFileTests המשתמשת ב-JUnit ובמחלקה TestDriver לביצוע בדיקות של Graph. המחלקה עוברת על כל הקבצים בתיקייה בה היא נמצאת ושולחת את כל הקבצים בעלי סיומת test, אחד אחרי השני, ל-TestDriver. המחלקה מכוונת את הקלט של TestDriver לקובץ בעל שם זהה וסיומת actual. ומשווה לקובץ התוצאות הנדרשות בעל שם זהה וסיומת expected.

עליכם לרשום קובצי קלט לבדיקות קופסה שחורה של המימוש של Graph. על כל קובצי הבדיקה להיות בעלי הסיומת test ולכל אחד מהם צריך להיות קובץ נוסף בעל שם זהה לשם קובץ הבדיקה ובעל הסיומת expected שיכיל את הפלט הצפוי. תנו לקובצי הבדיקה שתיצרו בסעיף זה שמות משמעותיים. נתונים מספר קובצי בדיקה לדוגמה.

נתונה המחלקה GraphTests היורשת מהמחלקה ScriptFileTests כדי לבצע בדיקות קופסה שחורה בהתאם לקובצי הקלט המתאימים. עליכם להוסיף למחלקה GraphTests בדיקות קופסה לבנה ל-Graph. יש לרשום בדיקות בעזרת תחביר JUnit רגיל ולא להשתמש בקובצי קלט, כמו שנעשה עבור בדיקות קופסה שחורה.

בסעיף זה אין צורך לבצע בדיקות של האלגוריתם למציאת מסלול קצר ביותר (שעדיין לא קיים בשלב זה).

להגשה ממוחשבת: קובצי הקלט לבדיקות קופסה שחורה שכתבתם ותוצאות הרצתם (קובצי test, expected ו-actual). כמו כן, המחלקות שכתבתם לבדיקות קופסה לבנה ותוצאות הרצתן.
להגשה "יבשה": תיעוד חיצוני המסביר את השיקולים בבחירת הבדיקות הנ"ל ומדוע הן מספקות.

ד.
במחלקות ScriptFileTests ו-TestDriver נעשה שימוש במחלקה java.nio.file.Path בשמה המלא. מדוע לא מתבצע לה import בכותרת המחלקות ולאחר מכן שימוש בשם המקוצר Path?

להגשה "יבשה": תשובה לשאלה הנ"ל.

שאלה 2 (25 נקודות)

בשאלה זו תממשו ותבדקו אלגוריתם למציאת מסלול קצר ביותר בגרף.

אלגוריתם למציאת מסלול קצר ביותר מנסה למזער פונקציית מחיר כלשהי הניתנת לחישוב על כל המסלולים בין נקודות התחלה (אחת או יותר) לנקודות סיום (אחת או יותר). ההפשטה הנתונה עבור מסלול בממשק Path מאפשרת, בין השאר, לקבל את מחירו של המסלול.

רב האלגוריתמים למציאת מסלול קצר ביותר פועלים על גרפים עם קשתות בעלות מחיר. במקרה שלנו, הצמתים הם בעלי מחיר ולא הקשתות. להלן מוצג אלגוריתם למציאת מסלול קצר ביותר במקרה זה. אלגוריתם זה הוא אלגוריתם חמדן (greedy) שהוא וריאציה של אלגוריתם Dijsktra.

האלגוריתם כתוב בפסבדו-קוד המשתמש בסימונים הבאים:

- [a, b, c] מייצג מסלול המורכב מסדרת הצמתים a, b ו-c.
- האופרטור + מציין שרשור. למשל: [a, b] + [c, d] = [a, b, c, d].
- בהינתן ש-p הוא אובייקט מטיפוס Path וש-m מציין פונקציה הממפה את p למחיר כשלהו, אזי m(p) מסמן את המחיר של p ב-m.

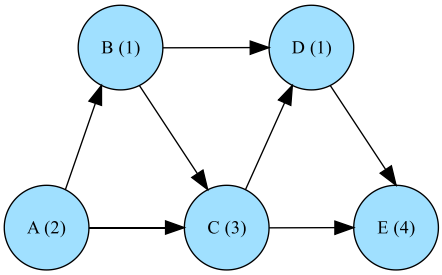
האלגוריתם הוא:

```
// Return a shortest path from any element of starts to any
// element of goals in a node-weighted graph.
Algorithm node-weighted-shortest-path(Set starts, Set goals) {

    // maps nodes -> paths
    Map paths = { forall start in starts | (start, [start]) }

    // The priority queue contains nodes with priority equal to the cost
    // of the shortest path to reach that node. Initially it contains the
    // start nodes
    PriorityQueue active = starts

    // The set of finished nodes are those for which we know the shortest
    // paths from starts and whose children we have already examined
    Set finished = { }
```

```
    while active is non-empty do {
        // queueMin is the element of active with shortest path
        queueMin = active.extractMin()
        queueMinPath = paths(queueMin)

        if (queueMin in goals) {
            return queueMinPath
        }

        // iterate over edges (queueMin, c) in queueMin.edges
        for each child c of queueMin {
            cpath = queueMinPath + [c]
            if (c not in finished) and (c not in active) {
                paths(c) = cpath
                insert c in active with priority equal to cpath's cost
            }
        }
        insert queueMin in finished
    }

    // execution reaches this point only if active becomes empty
    return No Path Exists
}
```

במהלך האלגוריתם נעשה שימוש בתור עדיפויות. זהו מבנה נתונים המאפשר להכניס איברים ולהוציא איבר בעל העדיפות הגבוהה ביותר (במקרה שלנו, ככל שמחיר המסלול נמוך יותר, כך עדיפותו גבוהה יותר). יש לשים לב כי המחלקה java.util.PriorityQueue לא מאפשרת להכניס באופן ישיר את העדיפויות של איבריה. במקום, היא משתמשת לצורך השוואת עדיפויות האיברים במתודה compareTo() של הממשק java.util.Comparable. הממשק הנתון Path מרחיב ממשק זה. לכן, עומדות בפניכם שתי אפשרויות למימוש: א. להכניס לתור העדיפויות מסלולים ולא צמתים. ב. לממש מחלקה עוטפת למחלקת צומת שתחזיק את אורך המסלול הקצר ביותר ותממש את java.util.Comparable בהתאם.

א.
נתון הגרף הבא:



הצמתים A, B, C, D ו-E הם בעלי מחיר 2, 1, 3, 1 ו-4 בהתאמה.

להגשה "יבשה": תיאור מילולי כללי של אופן פעולת האלגוריתם הנתון. בנוסף, הדגימו את שלבי פעולת האלגוריתם למציאת מסלול קצר ביותר בגרף הנתון מהצומת A לצומת E. בכל שלב יש לרשום מה מתבצע ואת ערכי paths, active ו-finished בסיומו.

ב.
עליכם לרשום מפרט עבור מחלקה בשם PathFinder ולממש מחלקה זו. המחלקה תכיל מתודה למציאת מסלול קצר ביותר בין שני צמתים בגרף. על המתודה לקבל אוסף של צמתי התחלה ואוסף של צמתי סיום ולמצוא מסלול קצר ביותר שהצומת הראשון בו הוא אחד מצמתי ההתחלה ושהצומת האחרון בו הוא אחד מצמתי הסיום.

על המחלקה PathFinder לתמוך בצמתים מטיפוס כלשהו. עליה לקבל את המחירים בעזרת הממשק Path. כדי ליצור חוסר תלות בטיפוס הצמתים, ניתן לתכנן את המתודה כך שתקבל מסלולי התחלה (מנוונים) ומסלולי סיום (מנוונים) במקום צמתי התחלה וצמתי סיום.

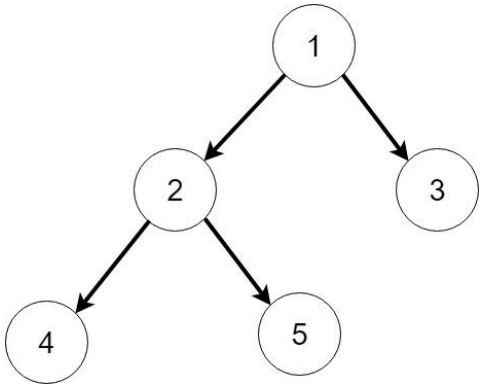
להגשה ממוחשבת: המחלקה PathFinder.

ג.
הרחיבו את סביבת הבדיקה שיצרתם בשאלה 1ג' עבור Graph כך שתבדוק גם את PathFinder ע"י הוספת קובצי קלט לבדיקות קופסה שחורה ובדיקות JUnit לבדיקות קופסה לבנה. לצורך בדיקת המימוש יש להשתמש במופעים של המחלקה הנתונה WeightedNodePath הממשת את הממשק Path והמייצגת מסלולים בגרף.

להגשה ממוחשבת: קובצי הקלט לבדיקות קופסה שחורה שכתבתם ותוצאות הרצתם (קובצי test, expected ו-actual). כמו כן, בדיקות קופסה לבנה שכתבתם ותוצאות הרצתן.
להגשה "יבשה": תיעוד חיצוני המסביר את השיקולים בבחירת הבדיקות הנ"ל ומדוע הן מספקות.

שאלה 3 (20 נקודות)

בשאלה זו נעסוק במבנה נתונים של עץ בינארי. עץ בינארי הוא גרף ללא מעגלים, שבו לכל צומת יש לכל היותר שני בנים. להלן איור של עץ בינארי לדוגמה בו כל צומת מסומן בעיגול וקיים חץ המצביע מכל צומת לצומת בן שלו.



נתון המימוש הבא למחלקה המייצגת צומת בעץ בינארי:

```
// A node in a binary tree. Each Node has left child node, right
// child node and an integer key value.
class Node {
   Node left, right;
   int key;

   public Node(int key){
     left = null;
     right = null;
     this.key = key;
  }
}
```

ונתון המימוש הבא עבור עץ בינארי העושה שימוש במחלקה Node:

```
class BinaryTree {
   Node root;             // root of the Tree

   public BinaryTree() {
     root = null;
   }

   public BinaryTree(int key) {
      root = new Node(key);
   }
}
```

הערה: עץ בינארי ריק הוא כזה שעבורו root == null.

1. מה הבעיה העיקרית במימוש המוצג בשאלה? כיצד ניתן לתקן בעיה זו (מבלי לפגוע בפונקציונאליות)? שים לב כי מדובר בבעיה שנמצאת בקוד הנתון ולא במתודות שחסרות.

2. סטודנט טען כי אם יימחק הבנאי הראשון (זה שלא מקבל פרמטרים) במחלקה BinaryTree, קטע קוד הקורא לבנאי זה בוודאות לא יושפע. האם הסטודנט צודק? נמק **בקצרה**.

3. נתונה המתודה הבאה של המחלקה BinaryTree המקבלת צומת node ומחליפה את הבן הימני של השורש של העץ ב-node:

```
public void setRightSonOfRoot(Node node) {
   root.right = node;
}
```

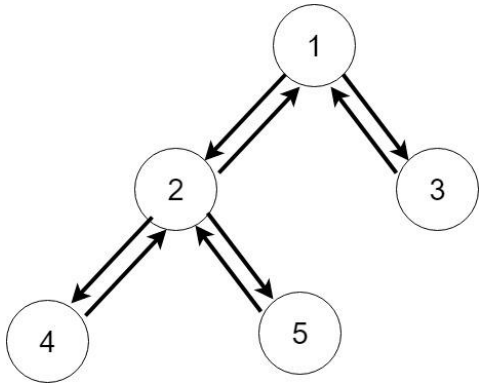
כתוב פסקאות requires, modifies ו-effects עבור מתודה זו. פסקת ה-requires צריכה להיות החזקה ביותר האפשרית (כלומר לאפשר קלט רחב ככל הניתן).

4. הוחלט לשנות את צמתי העץ כך שה-key שלהם יוכל להכיל אובייקט מטיפוס **מסוים** כלשהו ולא רק מספרים שלמים. אילו שינויים יש לערוך במחלקה Node על מנת לתמוך בשינוי המוצע? הסבר **בקצרה**. אין צורך לכתוב קוד.

5. נרצה לשנות את צמתי העץ מהסעיף הקודם כך שבנוסף לעובדה שהם יוכלו להכיל אובייקט מטיפוס מסוים כלשהו, נוכל להשוות בין שני אובייקטים מאותו הטיפוס. למשל, אם הטיפוס של ה-key הוא מחרוזת, נרצה לוודא שקיימת היכולת להשוות שתי מחרוזות כך שנדע איזו מחרוזת קודמת לשנייה בסדר לקסיקוגרפי. אילו שינויים יש לעשות עתה במחלקה Node? הסבר **בקצרה**. אין צורך לכתוב קוד.

6. כתוב representation invariant ו-abstraction function עבור המחלקות Node ו-BinaryTree.

7. עץ בינארי דו-כיווני הוא עץ בינארי שכל צומת שלו מצביע לצומת האב שלו. למשל, העץ הבא הוא עץ בינארי דו-כיווני:



כדי לממש עץ בינארי דו-כיווני, שינו את המחלקה Node באופן הבא (השינויים לעומת Node המקורי מודגשים):

```
// A node in a binary tree. Each Node has left child node, right
// child node, parent node and an integer key value.
class Node {
   Node left, right, parent;
   int key;

   public Node(int key, Node parent){
     left = null;
     right = null;
     this.parent = parent;
     this.key = key;
  }
}
```

הערה: ב-Node המייצג את שורש העץ מתקיים parent == null.

כתוב representation invariant עבור עץ בינארי דו-כיווני.

נספח: מבנה קובץ פקודות לבדיקה

קובץ בדיקה הוא קובץ טקסט המורכב מתווים אלפאנומריים והמכיל פקודה אחת בכל שורה. כל שורה מורכת ממילים המופרדות זו מזו ברווחים או בטאבים. המילה הראשונה בכל שורה היא שם הפקודה ושאר המילים הן ארגומנטים לפקודה זו. שורות המתחילות בסולמית (#) הן שורות הערה והן יועברו לאמצעי הפלט ללא שינוי בעת הרצת הבדיקה. שורות ריקות יגרמו להדפסת שורות ריקות באמצעי הפלט.

קובץ בדיקה מטפל בצמתים מטיפוס WeigthedNode ובמסלולים מטיפוס WeightedNodePath. לכל WeigthedNode יש שם ומחיר. לאחר שצומת כזה נוצר, ניתן להתייחס אליו בשמו, ובכל התייחסות אליו בפלט, הוא יוצג בשמו. גם לכל גרף יש שם, שההתייחסות אליו היא באותו אופן.

רשימת הפקודות והפלט שהן מחזירות היא:

CreateGraph *graphName*
יוצרת גרף חדש בשם *graphName*. הגרף מאותחל להיות ריק (ללא צמתים וללא קשתות). פלט הפקודה הוא:

created graph *graphName*

CreateNode *nodeName cost*
יוצרת צומת חדש בשם *nodeName* עם מחיר שהוא המספר השלם והלא שלילי *cost*. לאחר יצירת צומת בעזרת פקודה זו, ניתן להתייחס אליו בשמו. פלט הפקודה הוא:
created node *nodeName* with cost *cost*

AddNode *graphName nodeName*
מוסיפה את הצומת ששמו הוא המחרוזת *nodeName* לגרף ששמו *graphName*. פלט הפקודה הוא:

added node *nodeName* to *graphName*

AddEdge *graphName parentNode childNode*
יוצרת קשת בגרף *graphName* מהצומת *parentNode* לצומת *childNode*. פלט הפקודה הוא:

added edge from *parentNode* to *childNode* in *graphName*

ListNodes *graphName*
לפקודה זו אין השפעה על הגרף. פלט הפקודה מתחיל ב:

graphName contains:
ואחריו, באותה שורה, רשימה מופרדת ברווחים של שמות כל הצמתים בגרף *graphName*. על הצמתים להופיע בסדר אלפביתי. קיים רווח יחיד בין הנקודתיים לשם הצומת הראשון.

ListChildren *graphName parentNode*

לפקודה זו אין השפעה על הגרף. פלט הפקודה מתחיל ב:
the children of *parentNode* in *graphName* are:
ואחריו, באותה שורה, רשימה מופרדת ברווחים של שמות הצמתים שיש אליהם צומת מ-*parentNode* בגרף *graphName*. על הצמתים להופיע בסדר אלפביתי. קיים רווח יחיד בין הנקודתיים לשם הצומת הראשון.

FindPath *graphName from1* [*from2* [*from3* ...]] -> *to1* [*to2* [*to3* ...]]
הסימן -> מופרד משמות הצמתים ברווח משני צדדיו. הסוגריים המרובעים מציינים איברים שאינם חובה (הסוגריים המרובעים לא מופיעים בקובץ הקלט). ניתן לרשום מספר כלשהו של צמתי *from* ומספר כלשהו של צמתי *to*. לפקודה זו אין השפעה על הגרף. היא מוצאת ויוצרת פלט של מסלול קצר ביותר בין כל המסלולים האפשריים בין אחד מצמתי ה-*from* לאחד מצמתי ה-*to* בגרף *graphName*. פלט הפקודה מתחיל ב:
shortest path in *graphName*:
ואחריו, באותה שורה, רשימה מופרדת ברווחים של שמות הצמתים לפי סדר המעבר בהם מאחד מצמתי ה-*from* לאחר מצמתי ה-*to*. אם לא קיים מסלול, הפלט יהיה:
no path found in *graphName*

לדוגמה, עבור הקלט הבא:

```
CreateNode n1 5
CreateNode n3 1
CreateGraph A
AddNode A n1
CreateNode n2 10
AddNode A n2
CreateGraph B
ListNodes B
AddNode A n3
AddEdge A n3 n1
AddNode B n1
AddNode B n2
AddEdge B n2 n1
AddEdge A n1 n3
AddEdge A n1 n2
ListNodes A
ListChildren A n1
AddEdge A n3 n3
ListChildren A n3
FindPath A n3 -> n2
```

פלט תקין יהיה:

```
created node n1 with cost 5
created node n3 with cost 1
created graph A
added node n1 to A
```

```
created node n2 with cost 10
added node n2 to A
created graph B
B contains :
added node n3 to A
added edge from n3 to n1 in A
added node n1 to B
added node n2 to B
added edge from n2 to n1 in B
added edge from n1 to n3 in A
added edge from n1 to n2 in A
A contains: n1 n2 n3
the children of n1 in A are: n2 n3
added edge from n3 to n3 in A
the children of n3 in A are: n1 n3
shortest path in A: n3 n1 n2
```

הפלט עבור קלט לא תקין אינו מוגדר. ניתן להניח שהקלט הוא לפי התחביר החוקי בלבד.

אם הרצת הקלט גורמת לזריקת חריגה, תשלח הודעה מתאימה לאמצעי הפלט. יכולת זו כבר ממומשת בשלד של המחלקה TestDriver.