האוניברסיטה העברית בירושלים

בית הספר להנדסה ולמדעי המחשב ע"ש רחל וסלים בנין

סדנאות תכנות בשפת C ו־C++ – קיץ (קורס 67320)
C – תרגיל 2

תאריך ההגשה של התרגיל והבוחן התיאורטי: יום שלישי, ה־13 באוגוסט, 2019 – עד השעה 23:55;

הגשה מאוחרת (בהפחתת 10 נקודות): יום רביעי, ה־14 באוגוסט, 2019 – עד השעה 23:55.

<u>נושאי התרגיל</u>: מערכים, מצביעים, מחרוזות ופעולות על מחרוזות, structs, הקצאה דינמית וניהול זיכרון.

1 רקע

בקורסים קודמים למדתם על מבנה הנתונים **עץ**, הן במובן המתמטי והתיאורטי (במתמטיקה דיסקרטית ומבני נתונים) והן במובן התכנותי (במבוא למדעי המחשב ומבוא לתכנות מונחה עצמים). בתרגיל זה נממש את מבנה נתונים זה, ומספר אלגוריתמים שנוגעים לו בשפת C.

2 תזכורת ־ תורת הגרפים

להלן תובא תזכורת של מספר מונחים בסיסיים בתורת הגרפים.

- **הגדרה:** גרף מכוון (digraph) הוא גרף שבו ישנה משמעות לכיוונה של צלע, כלומר כל צלע יוצאת מצומת אחד ונכנסת לצומת אחר.
- **הגדרה**: יהי $G = \langle V, E \rangle$ גרף מכוון. מעגל מכוון הוא קבוצת קדקודים, $\{v_1, \ldots, v_n\} \in V$ כך שמתקיים $(v_1, v_2), \ldots, (v_{n-1}, v_n) \in E$.
- **הגדרה:** גרף רגולרי (Regular Graph) הוא גרף סופי שבו הדרגה של כל קדקוד היא מספר קבוע, $d \in \mathbb{N}$.
- **הגדרה**: יהי $G = \langle V, E \rangle$ גרף. G יקרא עץ אם ורק אם הוא גרף קשיר ללא מעגלים (DAG) כך ש־$|E| = |V| - 1$. לעץ יש "ענפים", דהיינו קשתות הגרף, ו"עלים", קרי הצמתים הקיצוניים.

- **מסקנה מההגדרה הקודמת:** $G = \langle V, E \rangle$ גרף (מכוון או לא מכוון). התנאים הבאים שקולים:
  - G הוא עץ – כלומר G הוא גרף קשיר ללא מעגלים פשוטים;
  - מתקיים $|E| = |V| - 1$ וגם ב־G אין מעגלים פשוטים;
  - מתקיים $|E| = |V| - 1$ וגם G קשיר.

בכל ההגדרות הבאות יהי $G = \langle V, E \rangle$ עץ מכוון:

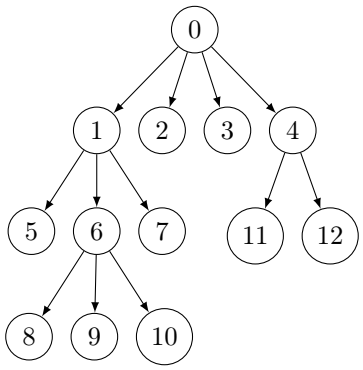
- **הגדרה: השורש** (root) של G יהא הצומת שממנו קיים מסלול לכל צומת אחר בגרף.
- **הגדרה**: **גובה העץ <u>המקסימלי</u>** הוא מספר **הקשתות** במסלול הארוך ביותר מהשורש לעלה.
- **הגדרה**: **גובה העץ <u>המינימלי</u>** הוא מספר **הקשתות** במסלול הקצר ביותר מהשורש לעלה.
- **הגדרה**: יהיו $u, v \in V$. u יקרא האב של v ו־v יקרא הבן של u אם ורק אם קיימת קשת בינן היוצאת מ־u ומסתיימת ב־v, כך שעומקו של u קטן ב־1 מעומקו של v. באופן שקול, מתקיים $(u, v) \in E$.
- **הגדרה**: עלה ב־G הוא כל צומת שאין לו בנים (כלומר צומת שהדרגה שלו היא 1).
- **הגדרה**: יהיו $u, v \in V$. u יקרא האב הקדמון של v, ובהתאמה v יקרא הצאצא הקדמון של u, אם ורק אם v הוא הבן של u ואם v הוא בן של צאצא של u
- **הגדרה:** גובה של צומת יהיה מספר הקשתות **במסלול הארוך ביותר** בין הצומת לאחד הצאצאים שלו. גובה של העץ יהיה הגובה של שורש העץ.

3 התוכנה TreeAnalyzer

בתרגיל זה נכתוב את התוכנית TreeAnalzyer. מטרת התוכנית שלנו היא לנתח עץ **עד־d־רגולרי,** שיתקבל כקלט מהמשתמש, ולהציג מספר תכונות שלו כפלט. בקצרה, התוכנית תקבל כארגומנטים קובץ txt שמתאר מבנה של **גרף מכוון** (לאו דווקא עץ) ו־2 קדקודים. התוכנית תאמת שאכן מדובר בעץ – ואם אכן זה המצב, היא תדפיס מידע על אודותיו ועל אודות שני הקדקודים שנקלטו, כדוגמת גובהו המינימלי והמקסימלי, כמות הקדקודים בעץ והמסלול הקצר ביותר מהקדקוד הראשון לשני.

3.1 תיאור העץ

הגדרה: עץ עד־d־רגולרי (**m-ary tree**) הוא עץ (כלומר גרף קשיר וחסר מעגלים) שבו הדרגה של כל קדקוד **אינה גבוהה מ־**d. במילים אחרות, אם $G = \langle V, E \rangle$ הוא העץ שבעניינו, אזי הוא יקיים $\forall v \in V \ deg(v) \leq d$. נוסף לכך, כל קדקוד בעץ מכיל מסומן על ידי מספר ייחודי ־ נקרא לו key. מפתח זה מקיים $key \in \mathbb{N}$.



איור 1: עץ עד־4 רגולרי (שימו לב לכמות הילדים של הקדקוד 1)

3.2 הקלט

התוכנית תקבל מהמשתמש בעת הרצתה שלושה ארגומנטים, דרך ה־cli: נתיב לקובץ txt המכיל מידע על **הגרף המכוון** ו־2 ערכים המסמנים קדקודים בגרף. אם כך, פורמט ההרצה יהיה:

```
$ ./TreeAnalyzer <Graph File Path> <First Vertex> <Second Vertex>
```

להלן מספר הערות הנוגעות לשלב זה:

- **לא ניתן** להניח שיתקבל מספר ארגומנטים תקין.
- **לא ניתן** להניח שהקובץ קיים (אך **אין צורך** להמיר relative path ל־absolute).
- **לא ניתן** להניח שהערכים שהתקבלו כקדקודים תקינים (בין אם כי לא נשלחו כלל, בין כי לא מדובר בערכים שמסמלים מספר תקין, בין כי לא קיים קדקוד כזה וכדומה).

3.3 פורמט קובץ הקלט

כאמור, הפרמטר הראשון שאותו התוכנית תקבל יהיה נתיב לקובץ txt שמכיל מידע על **גרף מכוון** שאותו נרצה לייצר. הקובץ בנוי בפורמט הבא:

- בתחילת הקובץ יופיע מספר אחד, נסמנו $n \in \mathbb{N} \cup \{0\}$, **שיסמן את כמות הקדקודים בגרף**.
- החל מהשורה השניה בקובץ, כל שורה תייצג קדקוד. **ערך הקדקוד** (ה־key) יקבע בסדר עולה החל מ־0 (0, 1, 2 וכך הלאה). במילים אחרות, כל קדקוד בגרף מיוצג על ידי השורה שבה הוא מופיע בקובץ **פחות 2** (כי השורה הראשונה מסמלת את כמות הקדקודים בגרף, וסופרים מ־0) , כך ש־$key(T[i-2]) = i-2$ לכל $2 \leq i \leq numberOfLines$ (מה שאומר ש־$i \in \mathbb{N} \cup \{0\}$). כך, השורה השניה תייצג את קדקוד מספר 0, השורה השלישית את קדקוד מספר 1 וכך הלאה והלאה.
- בכל שורה תופיע רשימה של מספרים (גודל הרשימה ≥ 0) שמופרדים ברווח. כל איבר שכזה מייצג **בן** של הקדקוד. לחלופין, אם זהו עלה, **כל התוכן של השורה** יהיה מקף (התו "–").

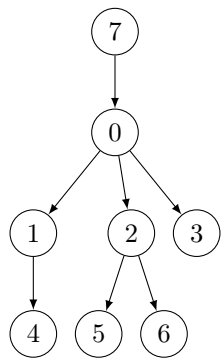
הערות:

- **ניתן** להניח שאורך כל שורה אינו עולה על 1024 תווים.
- **ניתן** להניח כי כל שורה מייצגת קדקוד וכי כל ערך מופרד ברווח (בודד).
- **לא ניתן** להניח שהקובץ אינו ריק. במקרה בו הקובץ ריק, יש להציג שגיאת קלט.
- **לא ניתן** להניח שהערכים שמופיעים בקובץ בהכרח תקינים.
  בפרט, להבדיל מהתרגיל הראשון, בתרגיל זה הנכם רשאים לעשות שימוש בפונקציות מובנות הממירות מחרוזת למספר, כדוגמת `strtod` ו־`strtol`. עם זאת, עליכם לוודא שאכן מדובר **אך ורק במספר ב־**$\mathbb{N} \cup \{0\}$ (למשל הקלט ''`1.2.3`'' אינו תקין).
- **לא ניתן** להניח שהגרף הנתון בקובץ הוא עץ (היזכרו בהגדרת "עץ" בחלק 2 לעיל).
- **לא ניתן** להניח שמספרי השורות מייצגים את **מיקומי** הקדקודים בגרף. כך למשל, **לא ניתן** להניח שהשורה השניה – שמייצגת את קדקוד מספר 0 – מייצגת את שורש העץ. ראו בהקשר זה את הדוגמה שלהלן, שם קדקוד 7 הוא האב של קדקוד 0 (והוא גם שורש העץ).
- **לא ניתן** להניח, כאמור לעיל, כי זוג הקדקודים שהתקבלו ב־cli אכן קיימים בגרף.
- **לא ניתן** להניח ש־n יהיה מספר תקין ועליכם לוודא זאת.
- **ניתן להניח** שכל בכל סוף קובץ **לא מופיע** התו ''`\n`''. **שימו לב:** ישנם הבדלים במימושים של פונקציות קריאת הקבצים במערכות הפעלה שונות (כן, גם אם הן מבוססות Unix). הבדלים אלו **קריטים** לשלב זה ועלולים להוביל לחוסר תאימות בהרצת תרגילכם במחשבי בית הספר ולמריטת שיערות רבה. לכן אנו חוזרים ומזכירים – עליכם לעבוד במחשבי האוניברסיטה ו**לוודא שפתרונכם עובד כמצופה במחשבים אלו.** פתרונות שעובדים כמו שצריך במערכות הפעלה אחרות, אך אינם עובדים כראוי תחת מחשבי בית הספר – יפסלו.

נביט בדוגמה הבאה: נניח שאנו קוראים קובץ טקסט המכיל את התוכן הבא (שימו לב ־ התחילית "`line #x:`" מסמנת את מספר השורה, למען בהירות הדוגמה, וכמובן שאינה מופיעה בקובץ. התוכן בכל שורה בקובץ הוא הטקסט שלאחר הנקודותיים):

```
line #1:8
line #2:1 2 3
line #3:4
line #4:5 6
line #5:-
line #6:-
line #7:-
line #8:-
line #9:0
```

אם כך, לאחר ניתוח הקובץ, נצטרך לקבל מבנה נתונים שמייצג את העץ הבא:



איור 2: עץ עד־4־רגולרי לדוגמה

3.4 המרת הקלט למבנה הנתונים המתאים

לאחר קריאת הקלט עליכם לאמת שאכן מדובר בעץ ולבנות ממנו מבנה נתונים מתאים. הנכם חופשיים לעצב מבנה או מבני הנתונים שישרתו אתכם בפתרון התרגיל כראות עיניכם, ובלבד שתעמדו בהנחיות הבאות:

- עליכם להשתמש **לכל הפחות** במבנה נתונים אחד (כלומר לכל הפחות ב־struct אחד).
- עליכם לעשות **לכל הפחות** שימוש אחד ב־`typedef`.
- עליכם לשמור את ערך הקדקוד (ה־key) בתוך מבנה הנתונים שיצרתם ולעשות בו שימוש.
- עליכם להשתמש בניהול זיכרון דינמי לצורך הקצאת וניהול **כל** מבנה נתונים (גם אם מדובר ביותר מאחד) שאתם יוצרים.
- **חשוב מאוד:** בכדי לפשט את התרגיל, החל מהרגע בו הגעתם למסקנה שמדובר בעץ – אין משמעות להיות הגרף מכוון. הווה אומר, תוכלו להתייחס להניח שכל קשת מכילה שני חצים (הווה אומר, האב מצביע לכל ילדיו וכל ילדיו מצביעים על האב).

3.5 טיפול בשגיאות בקלט ובבניית העץ

עליכם לטפל במקרים בהם לא התקבל קלט תקין. אם מספר הארגומנטים שנשלחו לתוכנה אינו תקין, עליכם להדפיס ל־`stderr` את הפלט:

```
Usage:  TreeAnalyzer <Graph File Path> <First Vertex> <Second
                    Vertex>\n
```

מנגד, אם נתקלתם בקלט שאינו עומד באחת דרישות התקינות (למשל הקובץ לא קיים או שאחד הקדקודים, u, v, לא קיימים), עליכם להדפיס ל־`stderr` את הפלט:

```
Invalid input\n
```

בשני המקרים "`\n`" מסמן ירידת שורה. לאחר הדפסת הפלט, בשני המקרים עליכם לסגור באופן מיידי את התוכנית עם קוד סיום `EXIT_FAILURE`. **שימו לב:** יש לתעדף שגיאות Invalid input ככול שהן רלבנטיות על פני שגיאות אחרות.

3.6 טיפול במקרה שבו מדובר בגרף שאינו עץ

אם הגרף שהתקבל **אינו עץ**, עליכם להדפיס את ל־stderr את השגיאה הבאה:

```
The given graph is not a tree\n
```

כש־ "\n" מסמן ירידת שורה. לאחר מכן, יש לסגור מיידית את התוכנית עם קוד שגיאה.

3.7 פלט התוכנית

אם הקלט תקין, על התוכנית שיצרתם להדפיס את הנתונים הבאים, **בסדר הופעתם שלהלן** (כל נתון יופיע בשורה חדשה, עם תחילית נתונה מראש, כפי שתראו בדוגמת הריצה שבהמשך):

- ערכו של קדקוד השורש (ה־key שלו);
- מספר הקדקודים שבעץ ומספר הצלעות שבעץ (כל אחד בשורה נפרדת);
- הגובה המינימלי בעץ והגובה המקסימלי בעץ (כל אחד בשורה נפרדת);
- קוטר העץ, קרי אורך המסלול הפשוט (שאין בו קדקודים כפולים) הארוך ביותר; שימו לב שאם העץ ריק אזי $D(T) = -1$ ואם לעץ קדקוד יחיד אזי $D(T) = 0$.

עתה, נזכיר שבנוסף לקובץ המתאר את הגרף, קיבלתם כקלט גם שני קדקודים, שנסמנם $u, v \in \mathbb{N}$ **בהתאמה לסדר קליטתם.** באשר אליהם, עליכם להדפיס את איברי המסלול הפשוט הקצר ביותר בין u לבין v – כשכל קדקוד במסלול יופרד ברווח. **שימו לב:**

- ישנה חשיבות לסדר בין u ל־v. הפיכת הסדר תוביל לאי תאימות עם ה־auto tests.
- **כאמור, מאחר שמדובר בעץ – עתה אין חשיבות לחצים ויהיו מקרים בהם תצטרכו, כחלק מריצת DFS ו־BFS, לסרוק בנוסף לילדים קדקוד – גם את ההורה שלו.**

3.8 הכוונה וחומר עזר

לשימושכם, לפתרון התרגיל עומד לרשותכם חומר העזר הבא:

- כנס פח לתרגיל מופיע תיאור של האלגוריתמים DFS ו־BFS. באמצעות DFS תוכלו לגלות האם קיים בגרף מעגל פשוט או לא. באמצעות BFS תוכלו לגלות מהו קוטר העץ וכן מהו את איברי המסלול הפשוט שבין u ל־v.
- באתר הקורס פורסם מימוש למבנה הנתונים "תור", המאחסן unsigned int. תוכלו להיעזר, אם תרצו, במבנה נתונים זה כדי לממש את תרגילכם – ובפרט את אלגוריתם BFS. כדי להשתמש בקבצים אלו, תדרשו להוסיף את הפקודה `#include ''queue.h''`. **שימו לב: חל איסור לערוך או להגיש את queue.h ו־queue.c.** הקבצים שפורסמו באתר הקורס יצורפו אוטומטית לפתרונכם בעת שיבחן על ידי הבדיקה האוטומטית.

4 דרישות זמן ריצה

על האלגוריתמים שתממשו במהלך התרגיל לעמוד בדרישות היעילות הבאות:

בדיקה האם זהו עץ $O(|V| + |E|)$

מציאת כמות הצלעות $O(1)$

מציאת שורש העץ $O(n)$

המסלול שבין u **ל-**v **וקוטר העץ** $O(n^2)$

5 דוגמה

נניח ש־g.txt קובץ עם התוכן שהוצג בפרק הקלט (פרק 3.2), אזי נוכל להריץ למשל:

```
$ ./TreeAnalyzer ./g.txt 4 3
Root Vertex: 7
Vertices Count: 8
Edges Count: 7
Length of Minimal Branch: 2
Length of Maximal Branch: 3
Diameter Length: 4
Shortest Path Between 4 and 3: 4 1 0 3
```

6 נהלי הגשה

- קראו בקפידה את הוראות תרגיל זה ואת ההנחיות להגשת תרגילים שבאתר הקורס.
- כתבו את כל ההודעות שבהוראות התרגיל בעצמכם. העתקת ההודעות מהקובץ עלולה להוסיף תווים מיותרים ולפגוע בבדיקה האוטומטית, המנקדת את עבודתכם.
- למען הסר ספק, **להבדיל מהתרגיל הראשון,** בתרגיל זה **לא קיים איסור** על שימוש בהקצאת זיכרון דינמית – ואתם מצופים לעשות שימוש בכלי זה. מנגד, גם בתרגיל זה חל איסור על שימוש ב־VLA. בנוסף, למען הסר ספק,בתרגיל זה **הנכם רשאים** לעשות שימוש בפונקציות מובנות הממירות מחרוזת למספר, כדוגמת `strtod, strtof` וכו'.
- פתרון בית הספר זמין בנתיב:

  `~proglab/www/c_ex2/TreeAnalyzer`

- עליכם ליצור קובץ tar הכולל **אך ורק** את הקובץ TreeAnalyzer.c וה־README שכתבתם (בפורמט הנדרש בהנחיות להגשת תרגילים). ניתן ליצור tar כדרוש על ידי:

  ```
  $ tar -cvf c_ex2.tar README TreeAnalyzer.c
  ```

  שימו לב: **אסור להגיש** את queue.h־ ו־queue.c. הם יתווספו אוטומטית.
  שימו לב: `TreeAnalyzer.c` יכלול את מלוא התרגיל, לרבות ה־main. על הקובץ להתקמפל כהלכה עם C99, כנדרש בהוראות להגשת תרגילים שפורסמו באתר הקורס.
- אנא וודאו כי התרגיל שלכם עובר את ה־Pre-submission Script **ללא שגיאות או אזהרות**. קובץ ה־Pre-submission Script זמין בנתיב.

  `~proglab/www/c_ex2/presubmission`

בהצלחה!!

7 נספח – אלגוריתמים שימושיים לפתרון התרגיל

בנספח זה נזכיר שני אלגוריתמים חשובים, שנלמדו בהרחבה בקורס מבני נתונים, ויכולים לסייע לכם בפתרון התרגיל. יהי $G = \langle V, E \rangle$ גרף מכוון.

7.1 אלגוריתם ה־DFS

כזכור, בתרגיל התבקשתם לקבל כקלט קובץ המייצג **גרף מכוון** ולנתח אותו **אך ורק** אם הוא עץ. הווה אומר **שאינכם יודעים האם הגרף המכוון שנקלט הוא עץ או לא.** אחת התכונות שגרף חייב לקיים כדי להיקרא עץ היא שלא יהיו בו מעגלים. לכן, בחלק זה נראה כיצד ניתן להשתמש באלגוריתם DFS כדי לזהות האם בגרף קיימים מעגלים או לא.
נתחיל מהסוף — מסקנה סופית: בגרף מכוון קיים מעגל אם"ם לאחר הרצת DFS (ללא תלות בקדקוד ההתחלה) נמצאה בגרף צלע־אחורה. דהיינו, קיים מעגל בגרף אם"ם:

$$\exists (u, v) \in E \quad S.t. \quad v.pre < u.pre < u.post < v.post$$

בקורס מבני נתונים למדתם על האלגוריתם לחיפוש עומק (Depth-First Search, או בקצרה – DFS), המאפשר לעבור על כל קדקודיו של גרף נתון באופן מבני ולהגיע לתובנות על הגרף. ניתן לממש את DFS, יחד עם תכונת ה־"שעון", בה נדון בהמשך, על ידי האלגוריתם הבא:

Algorithm 1 DFS

```
procedure DFS(G ⟨V, E⟩) :
  for each v ∈ G.V :
    v.visited ← False
  clock ← 0
  for each v ∈ G.V :
    if v.visited is False :
      explore(G, v)

procedure explore(G ⟨V, E⟩, v) :
  v.visited ← True
  preVisit(v)
  for each w such that (v, w) ∈ G.E :
    if w.visited is False :
      explore(G, w)
  postVisit(v)

procedure preVisit(v) :
  v.pre ← clock
  clock ← clock + 1

procedure postVisit(v) :
  v.post ← clock
  clock ← clock + 1
```

על קצה המזלג, זוהי שיטת הפעולה של DFS: האלגוריתם עובר באופן איטרטיבי על כל קודקודי הגרף. לכל קדקוד – DFS עובר על כל שכניו, באופן רקורסיבי. ברור ששיטה זו עלולה לגרום לכך שקדקודים "יבוקרו" בכפל. לכן, כל קודקוד שכבר טופל מסומן – ובכך מוודאים שלא עוברים עליו שוב אם מגיעים אליו דרך קודקוד נוסף. בדרך זו DFS עובר על מסלול ה־"מתרחק" מהקודקוד הראשון כמה שאפשר, ואחר כך חוזר על עקבותיו (backtrace) ופונה לעבר מסלולים נוספים, ה־"המרוחקים" מהקודקוד הראשון.
נשים לב שבשיטה זו DFS בעצם יוצר תת־יער של G: נגדיר גרף חדש, $G' = \langle V, E' \rangle$ המקיים $E' \subseteq E$, ובו נכלול אך ורק את הצלעות שבבדיקה שלהן בתוך הפונקציה explore, התגלה שהקודקוד השני עוד לא נחקר – כך שהאלגוריתם נדרש להיכנס לקריאה רקורסיבית על הקודקוד שני האמור. מדרך הגדרה זו נובע שב־G' לא יכול להיות מעגל.
עתה, יהיו $u, v \in V$, שניהם בתוך תת העץ G'. כדי לאבחן טוב יותר את הגרף, נסווג את היחסים ביניהם כך:

- **צלע־עץ (tree edge):** צלע הקיימת ב־E' ומחברת בין u לבין v.
- **צלע־קדימה (forward edge):** צלע שאינה ב־E', המחברת בין קודקוד לבין צאצא שלו בעץ **אחר** כלשהו ביער.
- **צלע־אחורה (backward edge):** צלע שאינה ב־E', המחברת בין קודקוד לבין אב קדמון שלו בעץ **אחר** כלשהו ביער.
- **צלע־חוצה (cross edge):** צלע שאינה ב־E', המחברת בין שני קדקודים בעצים שונים, או בין שני קודקודים בעץ כלשהו – כך שאף קדקוד איננו אב קדמון של השני.

כדי לגלות את מהו הסיווג של כל צלע, נבחן את היחסים של זמני ההגעה והיציאה אל הקדקודים ומהם. נעשה זאת כך: תחילה, נגדיר משתנה "שעון" גלובאלי לאלגוריתם (זהו המשתנה $clock$ שבאלגוריתם שלעיל), שיהווה מונה לכמות "הכניסות והיציאות" מהקדקודים. את מונה זה נגדיל ב־1 בכל פעם שניכנס לפונקציה explore ובאותו האופן נגדיל אותו ב־1 גם בכל פעם שנצא מ־explore. בהמשך, נוסיף שני שדות לכל קדקוד – השדה הראשון ייצג את "הזמן" שבו התחלנו בביקור ובטיפול בקדקוד מסויים (כלומר זמן "הכניסה" ל־explore) (זהו השדה pre שבאלגוריתם לעיל); השני, שדה שבו נשמור את "הזמן" שבו סיימנו לטפל בקדקוד (כלומר את "זמן היציאה" מ־explore) (זהו המשתנה $post$ באלגוריתם שלעיל).
עתה נראה את מספר טענות, שהוכחו בקורס במבני נתונים, ויכולות לשמש אותנו בתרגיל:

משפט: לאחר ריצת האלגוריתם DFS על הגרף G, לכל שני קודקודים $u, v \in V$, מתקיים בדיוק יחס אחד מה־3 הבאים:

1. הקטע $[v.pre, v.post]$ והקטע $[u.pre, u.post]$ זרים. במקרים אלו, u אינו צאצא של v ו־v אינו צאצא של u ביער ה־DFS.
2. הקטע $[v.pre, v.post]$ מוכל **ממש** בקטע $[u.pre, u.post]$. במקרים אלו, ו־v צאצא של u ביער ה־DFS.
3. הקטע $[u.pre, u.post]$ מוכל **ממש** בקטע $[v.pre, v.post]$. במקרים אלו, ו־u צאצא של v ביער ה־DFS.

טענה: תהי הצלע $(v, u) = e \in E$ בגרף המכוון G. ניתן לסווג את e בתוך יער DFS, לפי חותמות הזמן שלה, לפי הכללים הבאים:

1. e היא **צלע־עץ או צלע־קדימה** אם"ם $[v.pre, v.post] \subset [u.pre, u.post]$.

2. e היא **צלע־אחורה** אם"ם $[u.pre,\ u.post] \subset [v.pre,\ v.post]$.

3. e היא **צלע־חוצה** אם"ם $v.pre < v.post < u.pre < u.post$.

לפיכך – הטענה הסופית לה אנו נזקקים: בגרף מכוון קיים מעגל אם"ם יש בו צלע־אחורה לפי יער ה־DFS עליו ביצענו את הריצה, ללא תלות בסדר הריצה או בקדקוד ההתחלה.

7.2 אלגוריתם BFS:

עתה, נזכור שהתבקשתם למצוא את קוטר הגרף – כלומר את המסלול הפשוט הארוך ביותר בגרף, כמו גם את המסלול בין שני קדקודים שאת ערכיהם קלטתם כקלט ב־cli. **כדי לעשות זאת, תוכלו להשתמש באלגוריתם ה־BFS**, שאת תמציתו נביא להלן.
אלגוריתם חיפוש לרוחב (Breadth-First Search, או בקצרה – BFS) מאפשר לעבור על כל צומתי גרף בסריקה רוחבית. ניתן לתאר את האלגוריתם באמצעות ה־pseudo code הבא:

Algorithm 2 BFS

```
procedure BFS(G ⟨V, E⟩, s) :
  for each v ∈ V :
    v.dist = ∞
  s.dist ← 0
  s.prev ← NULL
  Q ← createQueue({s})
  while Q is not empty :
    u ← Q.dequeue()
    for each w such that (u, w) ∈ G.E :
      if w.dist is ∞ then :
        Q.enqueue(w)
        w.prev ← u
        w.dist ← u.dist + 1
```

על קצה המזלג, זוהי שיטת הפעולה של BFS: האלגוריתם משתמש במבנה הנתונים "תור" כדי לקבוע מהו הצומת הבא בו הוא עומד לבקר בכל פעם ש־BFS מבקר בצומת, הוא מסמן אותה כצומת שנבדקה בעבר (בדומה ל־DFS), ואז בודק את כל הצלעות שיוצאות מהצומת. אם צלע מובילה לצומת שטרם נבדק – צומת זה מתווסף לתור. בעזרת הליך זה אנו מבטיחים ש־BFS יסרוק את הצמתים בדיוק בסדר התואם למרחקם מהצומת ההתחלתי, שהרי צומת שנכנס לתור יצא ממנו רק לאחר שנוציא את כל הצמתים שהיו בו קודם. בשיטה זו BFS מאפשר למצוא את המסלול הקצר ביותר מקודקוד התחלתי כלשהו לשאר קדקודי הגרף.

שימו לב שההבדל המרכזי בין BFS לבין DFS הוא דרך סריקת הגרף. מהצד האחד, ב־DFS אנו סורקים כל קדקוד בגרף לעומק, דהיינו הסריקה מתבצעת על הבנים, הנכדים ושאר הצאצאים של קודקוד כלשהו, טרם עוברים לסרוק את שכניו. מנגד, ב־BFS אנו סורקים את הגרף לרוחב, דהיינו עבור כל קדקוד אנו בודקים קודם כל את **כל** הבנים שלו, לאחר מכן על את **כל** הבנים שלהם וכך הלאה.

לסיום, חשוב לציין שאת מבנה הנתונים "תור" מימשנו עבורכם והוא זמין באתר הקורס. תוכלו להסתמך על מבנה נתונים בזה בפתרון התרגיל, ובפרט במימוש BFS. **עם זאת, נזכיר שאינכם רשאים לבצע אף שינוי ב־queue.h ו־queue.c וממילא לאלו לא תהיה השפעה.**