

האוניברסיטה העברית בירושלים

בית הספר להנדסה ולמדעי המחשב ע"ש רחל וסלים בנין

## סדנאות תכנות בשפת C ו-C++ – קיץ (קורס 67312) C – תרגיל 2

**תאריך ההגשה של התרגיל והבוחן התיאורטי:** יום רביעי, ה-27 בנובמבר, 2019 – עד השעה 23:55;

**הגשה מאוחרת (בהפחתת 10 נקודות):** יום רביעי, 28 בנובמבר, 2019 – עד השעה 23:55.

**נושאי התרגיל:** מערכים, מצביעים, מחרוזות ופעולות על מחרוזות, structs, הקצאה דינמית וניהול זיכרון.

### 1 רקע

בקורסים קודמים למדתם על מבנה נתונים 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. שימו לב: תרגיל זה מכיל פרטים רבים. חשוב מאוד שתקראו את כל התרגיל במלואו טרם תתחילו לממשו.

### 2 תזכורת - תורת הגרפים

להלן תובא תזכורת של מספר מונחים בסיסיים בתורת הגרפים.

- **הגדרה:** גרף בלתי מכוון (UNDIRECTED GRAPH), ולעיתים בפשטות גרף הוא קבוצה של צמתים וקבוצה של קשתות (EDGES). כל קשת מקשרת בין שני צמתים.<sup>1</sup>
- **הגדרה:** גרף רגולרי (Regular Graph) הוא גרף סופי שבו הדרגה של כל קדקוד היא מספר קבוע,  $d \in \mathbb{N}$ .

---

<sup>1</sup>לפרטים נוספים: <http://bit.do/fhsxp>

• **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 (זהו בעצם משפט אוילר).

• **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  $G$  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 המקסימלי** הוא מספר הקשתות במסלול הארוך ביותר מהשורש לעלה.

• **הגדרה: גובה העץ המינימלי** 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  $u$ .

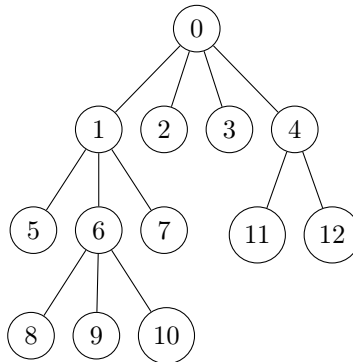
• **הגדרה:** גובה של צומת יהיה מספר הקשתות במסלול הארוך ביותר בין הצומת לאחד הצאצאים שלו. גובה של העץ יהיה הגובה של שורש העץ.

### 3 התוכנה TreeAnalyzer

בתרגיל זה נכתוב את התוכנית TreeAnalyzer. 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 לא מכוון ו-2 קדקודים. התוכנית, אך ורק אם הקלט תקין, תדפיס מידע על אודותיו (כפי שיתואר נרחבות בהמשך) 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 שבענייננו, אזי הוא יקיים  $\forall v \in V \deg(v) \leq d$ . נוסף לכך, כל קדקוד בעץ מסומן על ידי מספר ייחודי - נקרא לו  $key$ . מפתח זה מקיים  $key \in \mathbb{N}$ , זאת אומרת, המספר שמזהה את הקדקוד המדובר הינו מספר טבעי. על מנת להבין התבוננו בדוגמה הבאה:



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** (באופן כללי הגרף אינו מכוון, הקלט עצמו איתו נרכיב את העץ מכוון, זאת אומרת יש שורש יחיד והגדרה ברורה לאב ובן) 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.
- **ניתן** להניח שהקובץ קיים (**ואין צורך** להמיר 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 לא מכוון** שאותו נרצה לייצר. הקובץ בנוי בפורמט הבא:

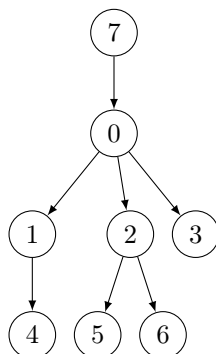
- בתחילת הקובץ יופיע מספר אחד, נסמנו  $n \in \mathbb{N} \cup \{0\}$ , **שיסמן את כמות הקדקודים בגרף**.
- החל מהשורה השניה בקובץ, כל שורה תייצג קדקוד. **עוד הקדקוד** (ה-`key`) יקבע בסדר עולה החל מ-0 (0, 1, 2 וכך הלאה). במילים אחרות, כל קדקוד בגרף מיוצג על ידי השורה שבה הוא מופיע בקובץ **פחות 2** (כי השורה 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  $0 \leq$ ) 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. בתרגיל זה הנכם רשאים לעשות שימוש בפונקציות מובנות הממירות מחרוזות למספר, כדוגמת `strtod` ו-`strtoul`. עם זאת, עליכם לוודא שאכן מדובר **אך ורק במספר** ב- $\mathbb{N} \cup \{0\}$  (למשל הקלט "1.2.3" אינו תקין).
  - עץ ריק, זאת אומרת  $n = 0$ , **אינו** תקין תחת הגדרת התרגיל.
  - **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 ש- $n$  הוא קלט שנמצא בשלמים (זה שהוא בשלמים לא בהכרח אומר שהוא תקין לפי דרישות התרגיל, שימו לב למעלה מתי  $n$  תקין).
  - **שימו לב:** ישנם הבדלים במימושים של פונקציות קריאת הקבצים במערכות הפעלה שונות (כן, גם אם הן מבוססות Unix). הבדלים אלו **קריטיים** לשלב זה ועלולים להוביל לחוסר תאימות בהרצת תרגילכם במחשבי בית הספר ולמריטת שיערות רבה. לכן אנו חוזרים ומזכירים – עליכם לעבוד במחשבי האוניברסיטה **ולוודא 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

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 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.

### 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,  $v$ ,  $u$ , לא קיימים), עליכם להדפיס ל-stderr את הפלט:

Invalid input\n

בשני המקרים "5" 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 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 (שאינו בו קדקודים כפולים) הארוך ביותר; שימו לב שאם 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 – והגרף לא מכונן יהיו מקרים בהם תצטרכו, כחלק מריצת BFS, לסרוק בנוסף לילדים קדקוד – גם את ההורה שלו.

### 3.7 הכוונה וחומר עזר

- לשימושכם, לפתרון התרגיל עומד לרשותכם חומר העזר הבא:
- כנס פח לתרגיל מופיע תיאור של האלגוריתמים BFS-`.BFS`. 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 לפתרוןכם בעת שיבחן על ידי הבדיקה האוטומטית.

## 4 דרישות זמן ריצה

על האלגוריתמים שתממשו במהלך התרגיל לעמוד בדרישות היעילות הבאות:

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, בתרגיל זה **הנכם רשאים (ואף נדרשים ומצופה מכם)** לעשות שימוש בהקצאת זיכרון דינמית. מנגד, גם בתרגיל זה חל איסור על שימוש ב-VLA.
- פתרון בית הספר זמין בנתיב:

`~labcc/www/c_ex2/TreeAnalyzer`

- עליכם לוודא שהקוד שלכם רץ באופן תקין וללא דליפות זכרון. לשם כך עליכם להשתמש בתוכנת VALGRIND. **תרגיל שיהיו לו דליפות זיכרון יאבד נקודות!**
- עליכם ליצור קובץ tar הכולל **אך ורק** את הקובץ TreeAnalyzer.c שכתבתם (בפורמט הנדרש בהנחיות להגשת תרגילים). ניתן ליצור tar כדרוש על ידי:

```
$ tar -cvf c_ex2.tar 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 אלגוריתם חשוב, שנלמד בהרחבה בקורס מבני נתונים, ויכולים לסייע לכם בפתרון התרגיל. יהי  $G = \langle V, E \rangle$  גרף לא מכוון.

### 7.1 אלגוריתם BFS:

נפתח בכך שנזכור 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 1 BFS

---

```
procedure BFS( $G \langle V, E \rangle, s$ ):  
  for each  $v \in V$ :  
     $v.dist = \infty$   
   $s.dist \leftarrow 0$   
   $s.prev \leftarrow \text{NULL}$   
   $Q \leftarrow \text{createQueue}(\{s\})$   
  while  $Q$  is not empty:  
     $u \leftarrow Q.dequeue()$   
    for each  $w$  such that  $(u, w) \in G.E$ :  
      if  $w.dist$  is  $\infty$  then:  
         $Q.enqueue(w)$   
         $w.prev \leftarrow u$   
         $w.dist \leftarrow u.dist + 1$ 
```

---

**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 (שגם אותו ראיתם בקורס "מבני נתונים") 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.**