הטכניון - מכון טכנולוגי לישראל
הפקולטה להנדסת חשמל
מבוא למערכות תוכנה

תרגיל בית 3 – ADT

מועד ההגשה:	יום ב', 26/12/2013 בשעה 23:59	
האחראי על התרגיל:	שיראל יוזף	shirel@campus

מטרה: שימוש ברשימות מקושרות גנריות, סטים גנריים וגרף על מנת לפתור בעיית ניווט.

רקע:

במסגרת תפקידך כראש צוות תוכנה בחברת robommt הוטל עליך ליצור מערכת חדשנית לניווט רובוטים.
אחת הדרכים האפשריות לתיאור מפה עבור דרכי ניווט הינה באמצעות גרף.
לאחר התייעצות עם צוות האלגוריתמיקה הוחלט שהגרף ימומש על ידי סט גנרי כיוון שמבנה נתונים זה מומש בעבר באמצעות רשימה מקושרת גנרית.
לצערנו, אחד הסטודנטים בחברה לא ביצע git push (אין צורך להשתמש בgit בקורס) וכל הקוד של הרשימה המקושרת הגנרית צריך להיכתב מחדש.

הערה חשובה 1: לאורך כל התרגיל יש לבדוק הקצאות זיכרון וארגומנטים אשר מועברים לפונקציות ולהחזיר ערך החזרה מתאים במקרה של כישלון. כישלון עלול להיות הקצאת זיכרון שנכשלה, פרמטרים לא תקינים, או כל כישלון אחר המוגדר עבור הפונקציה. הפכו את בדיקות אלו להרגל!
הערה חשובה 2: בדקו כל קוד שאתם כותבים בנפרד, כלומר, אם סיימתם לממש פונקציה עבור List כתבו קוד אשר בודק את הפונקציה. דיבוג של מערכת שלמה יהיה ארוך יותר.

חלק א':

יחידת הבניין הבסיסית לתרגיל זה תהיה ADT כללי מסוג List. ADT מסוג List הוא כזה ששומר איברים כאשר איבר יכול להופיע יותר מפעם אחת ברשימה.
בתרגיל זה מסופק לכם קובץ ה-h של מבנה הנתונים ועליכם לכתוב בעצמכם את קובץ list.c.
נשים לב כי הרשימה המקושרת אותה אנו מעוניינים לממש הינה גנרית ולא רשימה מקושרת של מספרים שלמים כפי שראינו בתרגול מספר 4.
לפני שמנסים לממש, כדאי לקרוא ולהבין מה פונקציות הממשק של הרשימה מבצעות על מנת לתכנן את המימוש בצורה נכונה.
על מנת לבצע חלק מהפעולות בList יהיה עלינו להשתמש באיטרטור. איטרטור יצביע על Node ברשימה המקושרת, ויאפשר לבצע מעבר איטרטיבי על הרשימה.
רמז למימוש: שימו לב להפרדה בין Node, לבין List, וחישבו באיזה struct עליכם להגדיר את האיטרטור, באיזה struct את האלמנט הגנרי ובאיזה struct את פונקציות המשתמש.

להלן תיאור אופן הפעולה של המתודות של List

ListCreate	מייצרת את הרשימה הריקה החדשה, מחזירה מצביעה לרשימה שנוצרה, או NULL במקרה של כישלון בפונקציה.
ListDestroy	מקבלת מצביעה לרשימה, הורסת את הרשימה, כולל מחיקת כל הPElem שהוכנסו בעזרת destroyFunc, וכל הNodes.
ListAdd	מעתיקה PElem בעזרת cloneFunc ומכניסה את ההעתק לתחילת הרשימה. הפונקציה מחזירה SUCCESS במקרה שהאלמנט הוכנס בהצלחה, או FAIL בכל מקרה אחר. האיטרטור לא חוקי לאחר פעולה זו,

	ולכן יש לעדכן אותו לערך NULL.
ListGetFirst	מעבירה את האיטרטור לראש הרשימה ומחזירה מצביע לPElem הנמצא בNode שאליו מצביע האיטרטור. במקרה של כישלון בפונקציה או רשימה ריקה יש להחזיר NULL.
ListGetNext	מקדמת את האיטרטור לחולייה הבאה ברשימה ומחזירה מצביע לPElem שנמצא בNode שאליו מצביע האיטרטור. אם הפונקציה נכשלה מוחזר NULL. אם האיטרטור לא תקין, ומצביע כרגע על NULL אז יוחזר NULL. אם הושג סוף הרשימה – מחזירה NULL.
ListGetSize	הפונקציה מחזירה int אשר מייצג את גודל הרשימה.
ListRemove	הפונקציה מקבלת מצביע לרשימה, ומוחקת את הNode שעליו האיטרטור המצביע. אם ההסרה הצליחה אז מוחזר SUCCESS והאיטרטור לא חוקי לאחר הפעולה, על כן יש לעדכן אותו לערך NULL. אם האיטרטור מצביע על NULL או שההסרה לא הצליחה יש להחזיר FAIL.

הקדמה חלק ב':
ADT מסוג Set הוא כזה ששומר איברים ללא חזרות. לדוגמא, נניח ויש לנו Set עם האיברים a, b, c ונוסיף לו את האיבר a אז נקבל שה Set לא השתנה. בתרגיל זה מסופקת לכם גרסה בסיסית של ADT זה ע"י קובץ c וקובץ h כאשר **המימוש של הSet מסתמך על מימוש הרשימה בחלק א'**.
בחלק ב' תממשו מבנה נתונים גרף על ידי שימוש בSet זה.

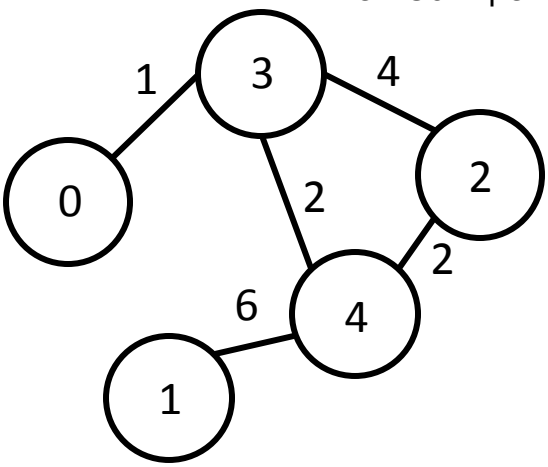
להלן תיאור אופן הפעולה של המתודות של Set:

SetCreate	מייצרת סט חדש ריק ומחזירה מצביע לסט מטיפוס PSet. אם הפונקציה נכשלה יוחזר NULL.
SetDestroy	הורסת את הסט, כולל מחיקת כל הפריטים שהוכנסו.
SetAdd	אם האיבר לא קיים בסט, הפונקציה מעתיקה את האיבר ומכניסה אותו לתוך הסט. אם בוצעה הכנסה יוחזר TRUE , בכל מצב אחר ואם האיבר קיים יוחזר FALSE.
SetFindElement	מקבלת PElem ובודקת האם קיים איבר זהה בסט. אם לא קיים איבר זהה בסט או אם הפונקציה נכשלה, יוחזר NULL, אחרת, יוחזר מצביע לאיבר בסט.
SetGetFirst	על מנת לבצע מעבר על איברי הסט יש לקרוא לפונקציה זו לפני הקריאות לSetGetNext. פונקציה זו תחזיר מצביע לאלמנט בסט, או NULL במקרה שהפונקציה נכשל או שהסט ריק.

SetGetNext	משמשת למעבר על איברי הסט, בכל קריאה יוחזר מצביע לאלמנט בסט. על מנת לבצע מעבר על איברי הסט יש לקרוא לSetGetFirst בפעם הראשונה כדי לקבל איבר בסט, ורק לאחר מכן, לבצע קריאות נוספות על ידי SetGetNext כדי לקבל את שאר האיברים בסט. כאשר בוצע מעבר על כל האיברים יוחזר NULL. אם הפונקציה נכשלה יוחזר NULL.
SetGetSize	במקרה שהסט ריק או שהתרחשה שגיאה יוחזר 0, אחרת יוחזר מספר האיברים בסט.
SetRemoveElement	מקבלת מצביע לסט ואלמנט שמעוניינים להסיר מהסט. אם האלמנט לא קיים בסט או שהתרחשה שגיאה, יוחזר FALSE, אחרת אם ההסרה הצליחה, יוחזר TRUE.

חלק ב':

גרף קשיר ממושקל לא מכוון מכיל Set של צמתים וSet של קשתות כאשר כל קשת מחברת בין שני צמתים ומכילה משקל. במקרה שלנו, לא יתכן כי קשת תחבר צומת לעצמו ולא יתכן כי צומת לא מחובר לאף קשת. יתכן כי מחוברות מספר קשתות לאותו צומת.
בסכימה הבאה למשל, הצמתים הינם עיגולים המזוהים על ידי מספרים שלמים מ-0 עד מספר הצמתים פחות אחד, וקשתות הן קווים המחברים בין צמתים, כאשר לכל קשת קיים משקל מספרי שלם.



במימוש שלנו, צומת מכיל int המייצג את זהות הצומת ומובטח שהצמתים ממוספרים בתחום $[0,n-1]$ כאשר n זה מספר הצמתים בגרף, וקשת מכילה מצביעים לעותקים פרטיים של שני הצמתים אותם היא מחברת. שימו לב כי אין חשיבות לסדר הצמתים הרשומים בקשת.
כלומר, הקשת 3-4 שקולה לקשת 4-3. (במקרה זה אם קשת 3-4 קיימת בסט, לא יהיה ניתן להכניס קשת 4-3).
בנוסף, משקל של קשת יהיה מספר שלם בתחום $[0,10]$.
לא יתכן כי קשת תחבר צומת לעצמו.
בתרגיל זה מסופק לכם קובץ ה-h של מבנה הנתונים המכיל את ההגדרות של מבני הנתונים Edge,Vertex.

שימו לב כי גרף אינו מבנה נתונים גנרי.

להלן תיאור אופן הפעולה של פונקציות הממשק של Graph:

GraphCreate	מייצרת גרף חדש ריק ומחזירה מצביע לגרף. במקרה כשלון יוחזר NULL.
GraphDestroy	הורסת את הגרף, כולל מחיקת כל הפריטים שהוכנסו.
GraphAddVertex	מקבלת מספר סידורי אשר מייצג את הצומת שמעוניינים להכניס. המספר **חייב** להיות n כאשר n הוא מספר הצמתים בגרף לפני הכנסה. לדוגמה, אם הגרף ריק, ניתן אך ורק להכניס את צומת מספר 0. אם בגרף יש צומת יחיד, ניתן להכניס צומת מספר 1. כמובן שלא יתכן שיכנסו שני צמתים בעלי אותו מספר סידורי. אם בוצעה הכנסה יוחזר True , בכל מצב אחר או אם הצומת קיים כבר, יוחזר False.
GraphAddEdge	מקבלת שני מספרים סידוריים של צמתים ואת משקל הקשת, ומכניסה את הקשת לתוך סט הקשתות. אם בוצעה הכנסה יוחזר True , בכל מצב אחר או אם הקשת קיימת כבר, יוחזר False. יש לשים לב לחוקים הנוגעים לקשתות שפורטו לעיל. כמובן שלא ניתן להכניס קשת המחוברת לצמתים שלא קיימים עדיין בגרף.
GraphNeighborVertices	פונקציה זו מקבלת מספר סידורי של צומת V ומחזירה אוסף חדש של **עותקים** של הצמתים המחוברים בקשת לצומת V. אם V לא קיים בגרף יוחזר NULL, אם התרחשה שגיאה בפונקציה יוחזר NULL. שימו לב: 1)אם אין צמתים שכנים, יוחזר אוסף ריק. 2) צומת אינו שכן של עצמו.
GraphGetNumberOfEdges	יוחזר מספר הקשתות בגרף, במקרה של שגיאה יוחזר 0.
GraphGetNumberOfVertices	יוחזר מספר הצמתים בגרף, במקרה של שגיאה יוחזר 0.
GraphVerticesStatus	פונקציה זו נוצרה על מנת שנוכל לוודא כי צמתים הוכנסו לגרף. על כן, פונקציה זו מקבלת מצביע לגרף ומחזירה **מצביע** לסט הצמתים בגרף. (אין ליצור סט חדש אלא להחזיר מצביע לסט הקיים)
GraphEdgesStatus	פונקציה זו נוצרה על מנת שנוכל לוודא כי קשתות הוכנסו לגרף. על כן, פונקציה זו מקבלת מצביע לגרף ומחזירה **מצביע** לסט הקשתות בגרף. (אין ליצור סט חדש אלא להחזיר מצביע לסט הקיים)

בנוסף לפונקציות אלו, יתכן ותצטרכו לממש פונקציות נוספות פרטיות (לא חשופות בממשק) בקובץ graph.c.

חלק ג':

כעת אתם מעוניינים לממש את פונקציית הממשק בגרף אשר מוצאת את **משקל** ואת המסלול הקצר ביותר מצומת מקור לכל שאר הצמתים בגרף בהינתן גרף קיים. כאשר במקרה שלנו, קצר משמעותו שסכום המשקלים על הקשתות הוא הקטן ביותר מבין כל המסלולים האפשריים. עליכם לממש את GraphFindShortestPath המוצהרת בקובץ graph.h.

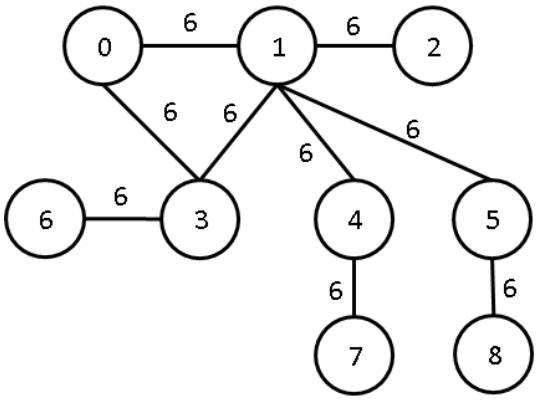
פונקציה זו מקבלת גרף, מספר סידורי של צומת מקור, מצביע למערך של מספרים שלמים dist, מצביע למערך של מספרים שלמים prev ומחזירה את משקל המסלול הקצר ביותר לצומת i מצומת המקור ב$dist[i]$ ואת הצומת הקודם לצומת i במסלול מצומת המקור ל-i ב $prev[i]$.

מערך dist ומערך prev מוקצים על ידי הקורא לפונקציה ובהם יש להכניס את פלט האלגוריתם לכן אין לשחררם בסוף הפונקציה. בקוד הבדיקה ההקצאה של מערכים אלו מסתמכת על GraphGetNumberOfVertices.

משקל המסלול הקצר ביותר מצומת לעצמו הינו 0 כלומר $dist[source]=0$ והצומת הקודם לו במסלול הוא הצומת עצמו כלומר $prev[source]=source$.

עבור כל הגרפים שעליהם תיבדק הפונקציה לא יתכנו מספר מסלולים קצרים ביותר עבור צומת המקור שיבחר לכל צומת אחר אלא רק מסלול קצר יחיד.

להלן דוגמה לגרף ולפלט **האלגוריתם (שימו לב הפונקציה לא מדפיסה לפלט הסטנדרטי)**:



המערך dist יכיל:

index	0	1	2	3	4	5	6	7	8
value	0	6	12	6	12	12	12	18	18

המערך prev יכיל:

index	0	1	2	3	4	5	6	7	8
value	0	0	1	0	1	1	3	4	5

קיימות מספר דרכים על מנת לפתור את סעיף זה, אבל צוות האלגוריתמיקה בחברת robommt יעץ לכם כי הדרך הפשוטה ביותר תהיה להשתמש באלגוריתם דייקסטרה.
פסאודו קוד של אלגוריתם דייקסטרה:
https://en.wikipedia.org/wiki/Dijkstra%27s_algorithm

```
function Dijkstra(Graph, source){

    create vertex set Q

    for each vertex v in Graph{     // Initialization
        dist[v] = INFINITY          // Unknown distance from source to v
        prev[v] = UNDEFINED         // Previous node in optimal path from source
        add v to Q                  // All nodes initially in Q (unvisited nodes)
    }
    dist[source] = 0                 // Distance from source to source
    prev[source] = source

    while Q is not empty{
        u = vertex in Q with min dist[u]     // Source node will be selected first
        remove u from Q

        for each neighbor v of u{            // where v is still in Q.
            alt = dist[u] + length(u, v)
            if (alt < dist[v]){              // A shorter path to v has been found
                dist[v] = alt
                prev[v] = u}
        }
    }
    return dist[], prev[]
}
```

טיפים למימוש:
1. במקום INFINITY יש להשתמש בINT_MAX זהו define שמוגדר בקובץ limits.h.
2. זיכרו שיש להוסיף עוד קוד על הפסאודו קוד, כמו בדיקת הצלחה של פרמטרים, יצירה של אובקייטים וכו...

makefile:

יש ליצור קובץ makefile כאשר הדגלים הנדרשים הם Wall– g– בלבד.
בנוסף, אין לבצע קישור של ספריות.

חלק 4 – פונקציית main – אין צורך לממש – חשוב לקרוא:

נתון לכם קובץ main.c שמכיל פונקציית main, מבצע ניתוח של הקלט ועונה על הדרישות הבאות: התוכנית קולטת שורה מהקלט הסטנדרטי ומבצעת את הפעולה המתאימה. אורך שורת הקלט המקסימלי הינו 256 תווים .

כל שורת קלט תהיה אחת מהצורות הבאות:

1. יצירת גרף חדש ריק

create_graph

2. הוספת צומת חדש על פי חוקי ההוספה של הגרף

add_vertex <vertex_serial_number>

3. הוספת קשת חדשה על פי חוקי ההוספה של הגרף

add_edge <vertex1_number> <vertex2_number> <weight>

4. הדפסת הגרף

print_graph

5. קבלת סט השכנים של צומת

get_neighbors <vertex_serial_number >

6. מציאת המסלול הקצר ביותר

find_shortest_path <source_vertex_serial_number >

7. יציאה מהתכנית (הגרף משוחרר מהזיכרון והתוכנית מסתיימת)

exit

<u>עבור</u> כישלון של פקודות **<u>ועבור</u>** פרמטרים לא תקינים למשל עבור ניסיון להכניס צומת לא לפי הסדר יש להדפיס הודעה בסגנון:

```
"add_vertex execution failed."
```

במקרה של שגיאה תודפס הודעת שגיאה ולהמשיך לפקודה הבאה.

<u>תרגילון bash:</u>

רשמו script יחיד בשם find_longest_word אשר מקבל כפרמטר של קובץ הנמצא באותה תיקייה בה נמצא הscript ומדפיס למסך את המילה הארוכה ביותר בקובץ ואת אורכה.
מובטח כי קיימת מילה ארוכה ביותר יחידה בקובץ.
הקובץ יכיל אך ורק אותיות, את התווים {:.,}(כלומר נקודה, פסיק ונקודתיים), ירידת שורה ויתכנו רווחים בין מילים (כלומר spaces).
ההדפסה למסך תהיה באופן הבא:

The longest word is encyclopedia and its length is 12.

אם הקובץ לא מכיל מילים יודפס:

The longest word is and its length is 0.

מצורפים קבצי פלט אליהם תוכלו להשוות את הפלט שלכם.
מובטח שהקובץ שמועבר כפרמטר לscript קיים בתיקייה.

הטכניון - מכון טכנולוגי לישראל
הפקולטה להנדסת חשמל
מבוא למערכות תוכנה

הנחיות הגשה:

1. קבצי קוד חלקיים, וכן קבצי קלט ופלט לדוגמה, נמצאים בתיקייה:
~eesoft/hmw/hmw3
לפני תחילת העבודה, הורידו את הקבצים לחשבונכם באמצעות הפקודה:
cp ~eesoft/hmw/hmw3/* .
2. עברו היטב על הוראות ההגשה של תרגילי הבית המופיעים באתר טרם ההגשה! ודאו כי התכנית שלכם עומדת בדרישות הבאות:
   - התכנית קריאה וברורה
   - התכנית מתועדת היטב לפי דרישות התיעוד המופיעות באתר
   - התכנית מתקמפלת ללא שגיאות וללא warnings כלל
   - התכנית רצה **ללא דליפות זיכרון** וגישות לא חוקיות לזיכרון כלל (בדיקה באמצעות valgrind)
   - התכנית נותנת פלט **זהה לחלוטין** לפלט הצפוי על כל קבצי הקלט שסופקו (בדיקה באמצעות פקודת diff על קבצי הפלט)
   - קובץ ה-makefile יוצר קובץ הרצה בשם הנדרש
3. יש להגיש קובץ tar יחיד המכיל את **כל הקבצים** שאתם נדרשים להגיש **ואותם בלבד** – ללא תתי-תיקיות. ודאו כי לא שכחתם את קובץ readme המכיל את פרטי הסטודנטים, וכן את ה-makefile במידה ונדרשתם.
4. שאלות בנוגע לתרגיל יש להפנות לפורום התרגיל ב-moodle בלבד – ניתן לשלוח שאלות במייל **למתרגל האחראי על התרגיל בלבד**, ורק במידה והשאלה מכילה פתרון חלקי.
5. סיכום מפרט התרגיל:

סעיף	**תיאור**		
נושא התרגיל	ADT		
תאריך ההגשה	יום ב', 26/12/2013 בשעה 23:59		
המתרגל האחראי על התרגיל	שיראל יוזף	shirel@campus	
תיקייה המכילה קבצים לשימוש הסטודנטים	~eesoft/hmw/hmw3		
קבצי הקוד הנתונים	list.h set.h set.c graph.h main.c		
קבצי הקלט והפלט הנתונים	err_1.txt	out_1.txt bash_out_1.txt bash_out_2.txt	in_1.txt
הקבצים שיש להגיש	readme makefile list.c graph.c find_longest_word		
שם תכנית ההרצה הדרושה (הנוצרת ע"י makefile)	robommt		
דגשים מיוחדים	...		

בהצלחה!