

מבני נתונים ויעילות אלגוריתמים

הוראות לנבחן

א. משך הבחינה: ארבע שעות.

ב. מבנה השאלון ומפתח ההערכה: בשאלון זה שני פרקים.

פרק ראשון	70	נקודות
פרק שני	30	נקודות
סה"כ	100	נקודות

ג. חומר עזר מותר לשימוש: כל חומר עזר כתוב בכתב-יד או מודפס על נייר.

ד. הוראות מיוחדות:

- את התשובות לשאלות שבפרק הראשון יש לכתוב במחברת הבחינה.
- את התשובות לשאלות שבפרק השני יש לסמן אך ורק על גבי דף התשובות שבנספח א'.
- ענה על מספר השאלות הנדרש בשאלון. המעריך יקרא ויעריך את מספר התשובות הנדרש בלבד, לפי סדר כתיבתן, ולא יתייחס לתשובות נוספות.
- לנוחותך, לשאלון זה מצורף מילון מונחים בשפות עברית, ערבית, אנגלית ורוסית. תוכל להיעזר בו בעת הצורך.

הוראות למשגיח:

בתום הבחינה יש לוודא שהנבחנים הדביקו את מדבקת הנבחן שלהם במקום המיועד לכך בדף התשובות שבנספח א' וצירפו אותו למחברת הבחינה.

כתוב במחברת הבחינה בלבד, בעמודים נפרדים, כל מה שברצונך לכתוב כטייטה (ראשי פרקים, חישובים וכדומה).
כתוב "טייטה" בראש כל עמוד טייטה. כתיבת טייטות כלשהן על דפים שמחוץ למחברת הבחינה עלולה לגרום לפסילת הבחינה!

בשאלון זה 42 עמודים ו-3 עמודי נספחים.

ההנחיות בשאלון זה מנוסחות בלשון זכר,
אך מכוונות הן לנבחנות והן לנבחנים.

השאלות

פרק ראשון (70 נקודות)

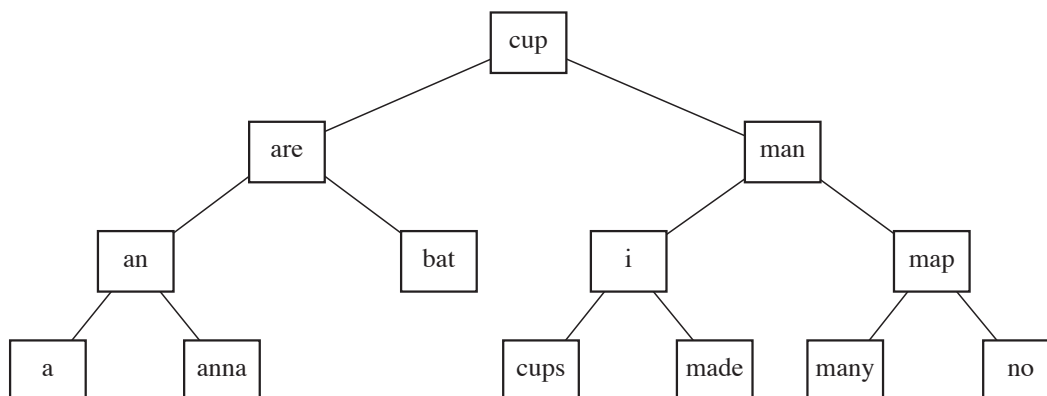
ענה על השאלות 1-2 – שאלות חובה.

שאלה 1 – שאלת חובה (40 נקודות)

בשאלה זו עשרה סעיפים (א'–י'). עליך לענות על כל הסעיפים א'–ז', ועל שני סעיפים נוספים מבין הסעיפים ח'–י'.

נדרש לאחסן אוסף של מחרוזות הממוינות בסדר מילוני, ולבצע פעולות, כגון הכנסת מחרוזת, מחיקת מחרוזת וחיפוש מחרוזת, בסיבוכיות זמן יעילה. לשם כך, הוחלט להשתמש במבנה הנתונים "עץ חיפוש בינארי של מחרוזות". להלן דוגמה ל"עץ חיפוש בינארי של מחרוזות", שבצמתיו מאוחסנות 13 מחרוזות שהוכנסו לעץ בזו אחר זו בסדר שלהלן (קרא משמאל לימין):

"cup", "are", "man", "bat", "i", "cups", "map", "an", "made", "a", "no", "many", "anna"



איור א' לשאלה 1

סריקה תוכית (inorder) של העץ תאפשר הצגה של המחרוזות בסדר מילוני, וכך יתקבלו המחרוזות בסדר שלהלן (קרא משמאל לימין):

"a", "an", "anna", "are", "bat", "cup", "cups", "i", "made", "man", "many", "map", "no"

כעת, נדרש להציע מבנה נתונים **אחר**, אשר יתמוך בכל הדרישות שלעיל. מבנה הנתונים החדש יכלול את תכונותיו של עץ החיפוש הבינארי, אך יהיה גם חסכוני מבחינת שטח האחסון, ויאפשר פעולות חיפוש מיוחדות שיוצגו בהמשך. לשם כך, נבחר מבנה הנתונים: "**עץ חיפוש טרינארי**" (TST - Ternary Search Tree).

הנחות יסוד:

1. המחרוזות שיאוחסנו בעץ יהיו מורכבות רק מאותיות קטנות מתוך האלף-בית האנגלי.
2. בסעיפים שבהם נדרש לחשב את סיבוכיות זמן הריצה, אפשר להניח כי העץ **מאוזן**, וכן ש- N מציין את מספר המחרוזות בעץ.

"עץ חיפוש טרינארי" (TST - Ternary Search Tree) הוא מבנה נתונים המשלב את תכונותיהם של העץ הטרינארי ועץ החיפוש, באופן הזה:

עץ טרינארי – לכל צומת בעץ TST יש שלושה בנים לכל היותר: שמאלי, ימני ואמצעי.

עץ חיפוש – הערך השמור בכל צומת עץ TST גדול מכל ערך שנמצא בתת-העץ השמאלי של הצומת, וקטן מכל ערך שנמצא בתת-העץ הימני של הצומת – כמו בעץ החיפוש הבינארי שתואר באיור א' לעיל.

עם זאת, בעץ החיפוש הבינארי שתואר מאחסנים את כל תווי המחרוזות בצומת אחד, ואילו בעץ TST כל תו מתווי המחרוזות מאוחסן בצומת נפרד בעץ, כמתואר להלן:

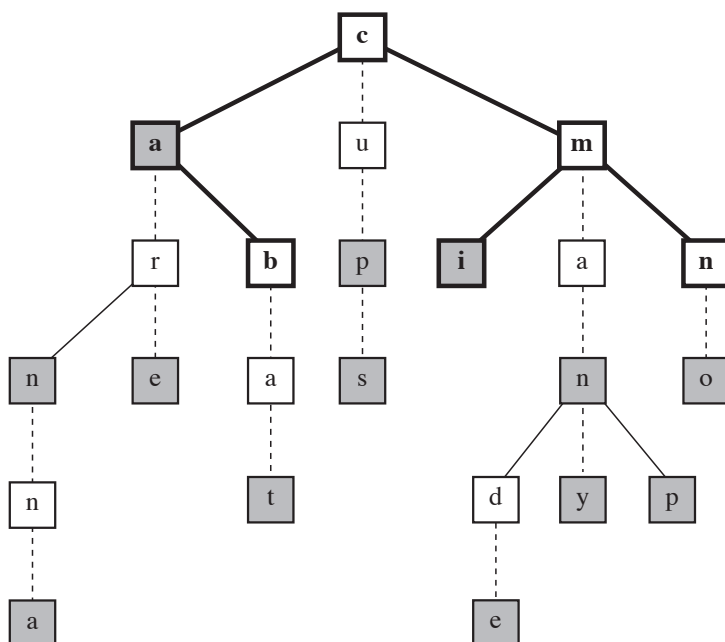
התו הראשון של המחרוזת יימצא בשורש העץ, או בצומת אחר שניתן להגיע אליו משורש העץ, תוך כדי מעבר דרך בנים שמאליים או ימניים **בלבד**, והוא יכונה "צומת תחילת-המחרוזת".

התווים הבאים של המחרוזת יימצאו רק בתת-העץ שעליו מצביע הבן האמצעי של צומת תחילת-המחרוזת.

התו האחרון של המחרוזת יימצא בצומת כלשהו, אשר יכיל סימן מיוחד המציין את סוף המחרוזת, והוא יכונה "צומת סוף-המחרוזת".

להלן איור של עץ TST שבו מאוחסנות כל 13 המחרוזות שאוחסנו בעץ החיפוש הבינארי שלעיל:

"cup", "are", "man", "bat", "i", "cups", "map", "an", "made", "a", "no", "many", "anna"



איור ב' לשאלה 1

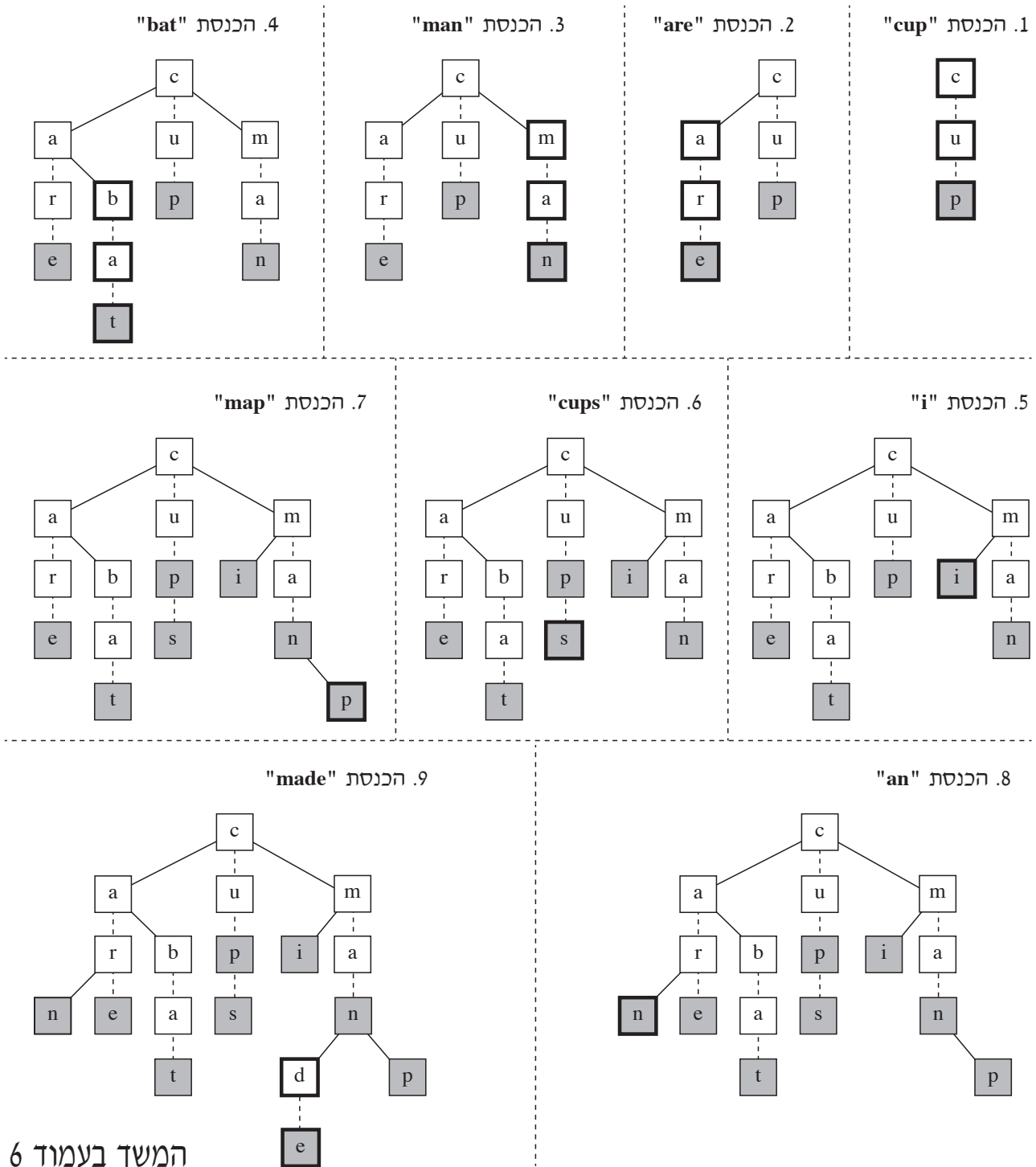
שים לב:

1. צמתים הצבועים באפור מציינים את סוף המחרוזות (ישנם 13 כאלה), וצמתים המסומנים במסגרת ריבועית מודגשת מציינים תחילת מחרוזות (ישנם 6 כאלה).
2. הצומת שמכיל את התו m הוא צומת תחילת מחרוזות, ומסומן, כאמור, במסגרת ריבועית מודגשת. בנו האמצעי של הצומת הזה מצביע לתת-עץ TST שבו נמצאות כל המחרוזות בעץ שמתחילות בתו m.
3. למחרוזות: "man", "many", "made", "map", יש תחילית משותפת "ma", והן חולקות צמתים משותפים בעץ, החוסכים מקום אחסון.
4. צומת המציין סוף מחרוזות אינו חייב להיות עלה. לדוגמה: הצומת האפור, השמאלי ביותר, המכיל את התו n, משמש כצומת רגיל בעבור המחרוזות anna, אך הוא מציין גם את סוף המחרוזות בעבור המחרוזות an.

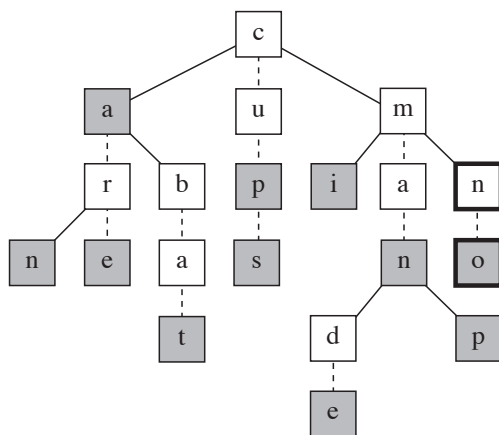
להלן שלבי בניית עץ ה-TST המוצג באיור ב' לשאלה, והכנסת 13 המחרוזות שלהלן בזו אחר זו (קרא משמאל לימין):

"cup", "are", "man", "bat", "i", "cups", "map", "an", "made", "a", "no", "many", "anna"

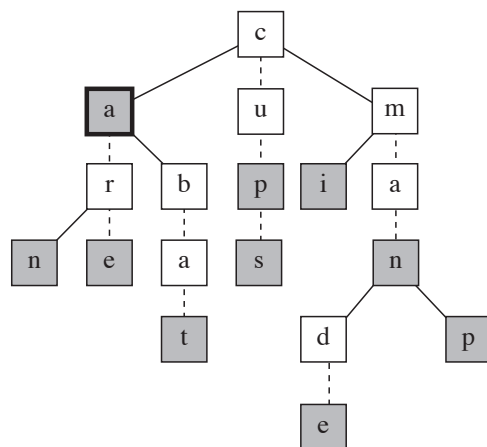
שים לב: בכל שלב יודגשו צומת שנוסף לעץ בשלב הנוכחי ו/או צומת שנעשה בו עדכון.



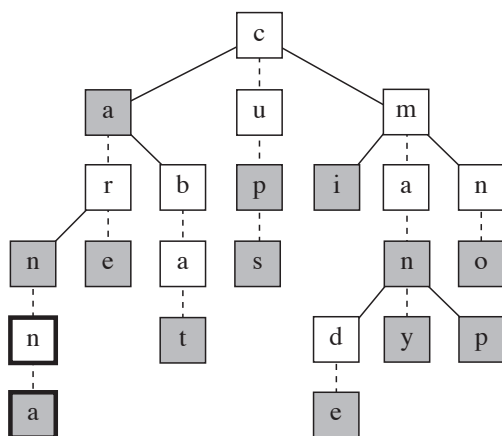
11. הכנסת "no"



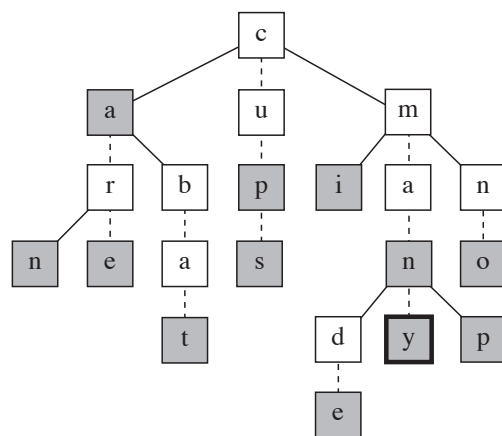
10. הכנסת "a"



13. הכנסת "anna"



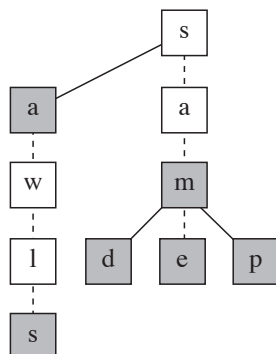
12. הכנסת "many"



איור ג' לשאלה 1

ענה על כל הסעיפים א'-ז'.

(5 נק') א. נתון עץ ה-TST שלהלן:



איור ד' לשאלה 1

כתוב במחברתך (משמאל לימין) את כל המחרוזות המאוחסנות בעץ הזה, **בסדר מילוני**.

(5 נק') ב. לעץ ה-TST הנתון באיור ד' הוכנסו חמש מחרוזות **נוספות** כדלהלן (סדר ההכנסה הוא משמאל לימין):

"sim", "be", "the", "s"

צייר במחברתך את עץ ה-TST המתקבל לאחר הכנסת חמש המחרוזות הללו, וסמן את הצמתים המציינים סוף מחרוזות.

להלן הגדרת הטיפוסים והמבנה של **צומת** בעץ TST בשפת C:

```
typedef enum {FALSE, TRUE} boolean;
```

```
typedef struct tstType
```

```
{
```

```
    char ch; // תו (אות קטנה מתוך האלף-בית האנגלי) המאוחסן בצומת
```

```
    boolean isLast; // משתנה בוליאני המציין אם הצומת מהווה סוף מחרוזת
```

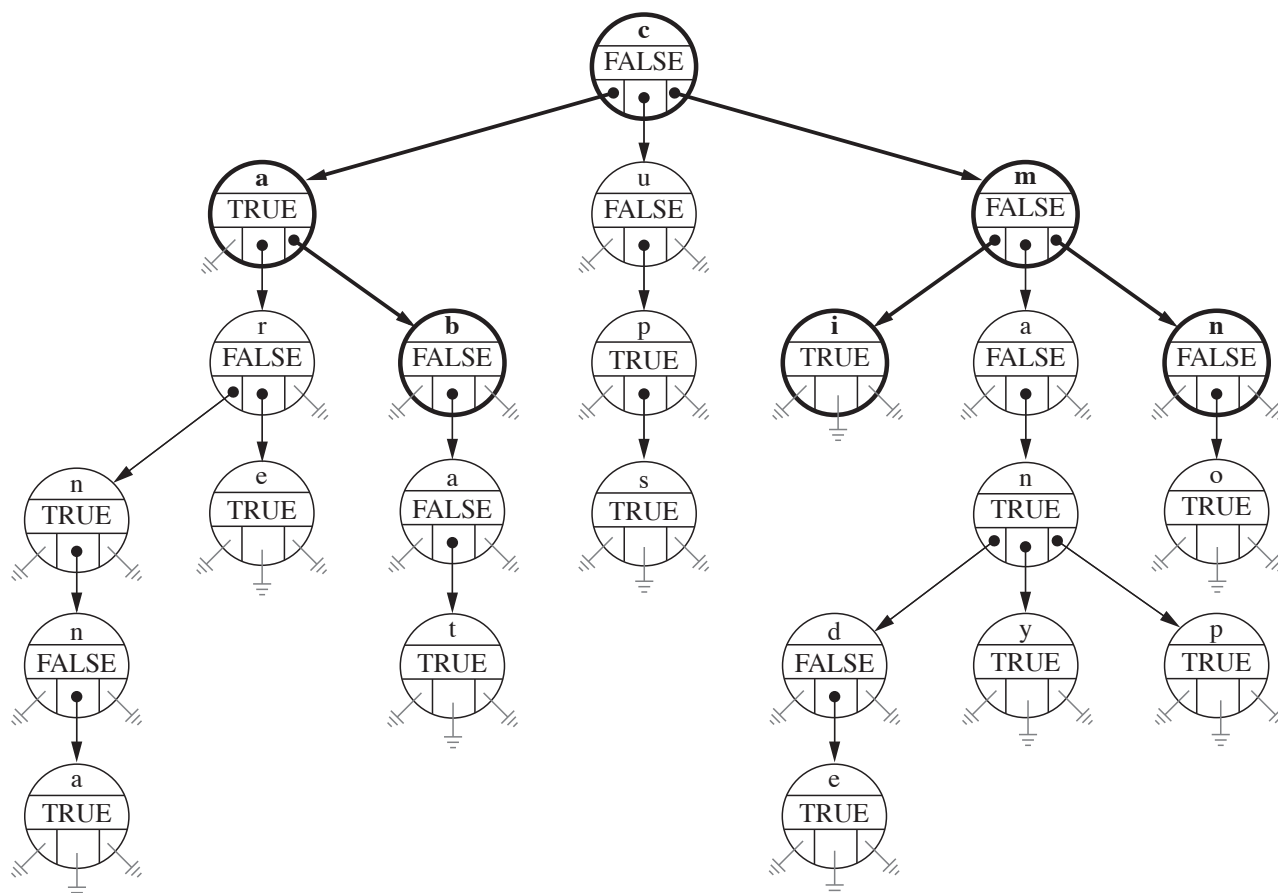
```
    struct tstType *left; // מצביע לתת-עץ שמאלי שבו מחרוזות המתחילות בתו הקטן מ-ch
```

```
    struct tstType *mid; // מצביע לתת-עץ אמצעי שבו מחרוזות המתחילות בתו בדיוק ch
```

```
    struct tstType *right; // מצביע לתת-עץ ימני שבו מחרוזות המתחילות בתו הגדול מ-ch
```

```
} tstRec, *tstPtr;
```

להלן תיאור סכמתי של עץ ה-TST הנתון באיור ב' (בעמוד 4 לעיל), על-פי הייצוג הנ"ל:



איור ה' לשאלה 1

ג. (6 נק') לפי פונקציית רקורסיבית שכותרתה:

```
void insert(tstPtr *root, char *str)
```

פונקצייה זו מקבלת את הפרמטרים האלה:

root – כתובת המצביע לשורשו של עץ TST

str – מחרוזת לא ריקה, שיש להכניסה לעץ

פונקצייה זו מטפלת בהכנסת המחרוזת str לעץ ה-TST.

בפונקצייה חסרים חמישה ביטויים המסומנים במספרים בין סוגריים עגולים. כתוב במחברת הבחינה את מספרי הביטויים החסרים (1)–(5) בסדר עולה, וכתוב ליד כל מספר את הביטוי החסר שהוא מייצג.

```
void insert(tstPtr *root, char *str)
{
    if (!(*root))
    {
        *root = (tstPtr) malloc(sizeof(tstRec));
        (*root)->ch = *str;
        (*root)->isLast = FALSE;
        (*root)->left = (*root)->mid = (*root)->right = NULL;
    }
    if (*str < (*root)->ch)
        insert(_____(1)_____, *str);
    else
    {
        if (_____(2)_____)
            insert(&((*root)->right), _____(3)_____);
        else
        {
            if (*(str+1))
                _____(4)_____;
            else
                _____(5)_____;
        }
    }
}
```

ד. (2 נק') מהי סיבוכיות זמן הריצה של הפונקצייה insert , כפונקצייה של N , אם ידוע גם ש-M הוא אורך המחרוזת המיועדת להכנסה?

ה. (6 נק') לפניך פונקצייה **רקורסיבית** שכותרתה:

```
boolean search(tstPtr root, char *str)
```

פונקצייה זו מקבלת את הפרמטרים האלה:

root – מצביע לשורשו של עץ TST

str – מחרוזת לא ריקה, שאותה מעוניינים לחפש בעץ

פונקצייה זו בודקת אם המחרוזת str מאוחסנת בעץ שלשורשו מצביע root .

אם כן – הפונקצייה מחזירה TRUE , אחרת – היא מחזירה FALSE .

חיפוש המחרוזת מתבצע באופן הזה:

מתחילים לסרוק את העץ החל משורש העץ, במטרה לאתר בעץ צומת תחילת-מחרוזת המכיל תו **זהה** לתו הראשון במחרוזת str . מתקדמים רק שמאלה או ימינה עד למציאת התו, כמו בחיפוש בעץ חיפוש בינארי.

משנמצא הצומת המתאים, ממשיכים את הסריקה דרך תת-העץ **האמצעי** של הצומת הזה, ומחפשים שם את התו **השני** במחרוזת str . כך הלאה עבור כל תו ותו במחרוזת str , עד אשר מגיעים לצומת שבו נמצא התו האחרון במחרוזת str וגם ערך המשתנה isLast שבו הוא TRUE .

אם הסריקה מסתיימת בצומת שבו ערך המשתנה isLast הוא FALSE **או** שהגענו אל מחוץ לעץ (NULL), סימן שמחרוזת החיפוש אינה מאוחסנת בעץ, והפונקצייה כאמור מחזירה FALSE , אחרת – הפונקצייה מחזירה TRUE .

בפונקצייה חסרים **חמישה** ביטויים המסומנים במספרים בין סוגריים עגולים. כתוב במחברת הבחינה את מספרי הביטויים החסרים (1)–(5) בסדר עולה, וכתוב ליד כל מספר את הביטוי החסר שהוא מייצג.

```
boolean search(tstPtr root, char *str)
{
    if (!root)
        return FALSE;
    if (*str < root->ch)
        return _____ (1) _____;
    if (_____ (2) _____)
        return search(root->right, str);
    if (_____ (3) _____)
        _____ (4) _____;
    return search(_____ (5) _____);
}
```

(2 נק') 1. מהי סיבוכיות זמן הריצה של הפונקצייה `search`, כפונקצייה של N , אם ידוע גם ש- M הוא אורך המחרוזת לחיפוש?

(2 נק') 2. האם תשתנה סיבוכיות זמן הריצה של הפונקצייה `search` אם הפונקצייה תמומש באופן איטרטיבי? ענה "כן" או "לא", ונמק בקצרה.

בספרייה הלאומית הוחלט להקים מערכת ממוחשבת שתאפשר לקוראים לחפש מילה כלשהי בספר, ולאתר את מספרי העמודים שבהם היא נמצאת.

כדי לאפשר חיפוש יעיל ומהיר, הוחלט ליצור לכל ספר בספרייה **אינדקס מילים** שיכלול את כל המילים בספר, וליד כל מילה תובא רשימת כל העמודים שבהם היא מופיעה.

המילים באינדקס יהיו ממוינות בסדר מילוני, ורשימת העמודים המופיעה ליד כל מילה תהיה ממוינת בסדר עולה על-פי מספר העמוד.

לדוגמה, נניח שבספרייה קיים ספר שבו מופיעות, בין היתר, המילים האלה:

"a", "an", "anna", "are", "bat", "cup", "cups", "i", "made", "man", "many", "map", "no"

להלן, **אינדקס המילים** (החלקי) של ספר זה:

a	2, 3, 5, 6, 8, 10, ...
an	1, 2, 3, 5, 7, 11, ...
anna	9, 30, 60, 65, 117, 120, 750
are	1, 3, 4, 5, 6, 9, ...
bat	10, 34 , 77, 300
cup	34 , 97, 222, 223, 400, 502
cups	100, 102, 301, 375, 655
i	1, 2, 3, 4, 5, 6, ...
made	21, 34 , 49, 55, ...
man	6, 7, 10, 40, ...
many	1, 3, 12, 87, ...
map	58, 66, 70, 128, ...
no	2, 3, 5, 11, ...
:	

ניתן לראות למשל כי:

1. המילה `bat` מופיעה בספר בארבעת העמודים האלה: 10, 34, 77, 300.

2. בעמוד 34 מופיעות המילים: `bat`, `cup`, `made` (ואולי עוד מילים).

הוחלט לייצג את **אינדקס המילים** על-ידי עץ TST, כך שהמחרוזות המאוחסנות בעץ יהיו המילים באינדקס, ועבור כל מילה כזו תישמר **רשימה מקושרת** של העמודים שבהם מופיעה המילה הזו בספר.

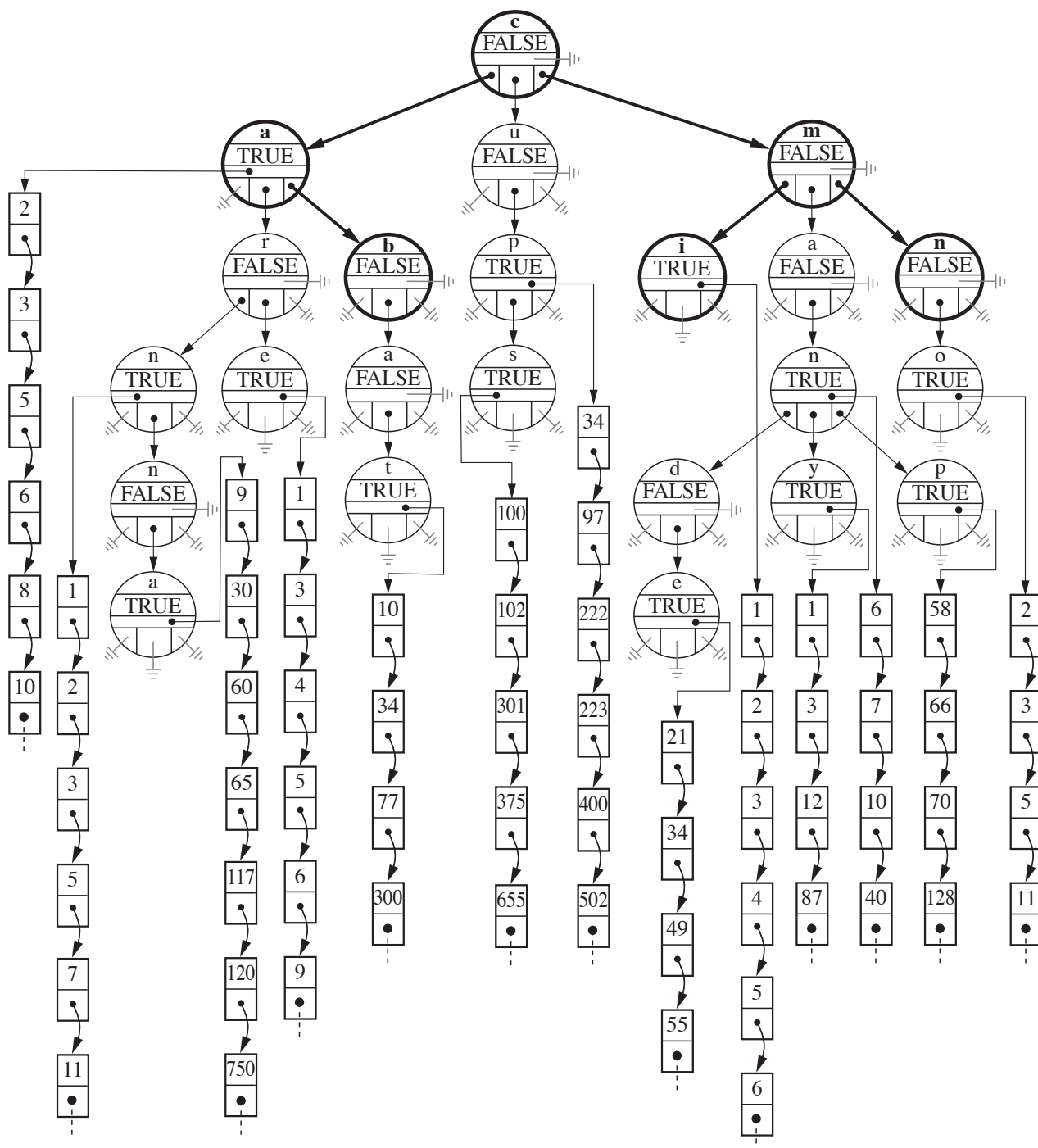
להלן הגדרת הטיפוסים והמבנה **המעודכן** של צומת TST, המתאים לייצוג **אינדקס המילים**:

```
typedef struct listType          // הגדרת צומת ברשימה המקושרת של העמודים
{
    int pnum;                    // מספר עמוד
    struct listType *next;       // מצביע לאיבר הבא ברשימה
} *listPtr;

typedef struct tstType           // הגדרת צומת של אות במילה השמורה באינדקס המילים
{
    char ch;                     // אות קטנה מתוך האלף-בית האנגלי המאוחסנת בצומת
    boolean isLast;              // משתנה בוליאני המציין אם הצומת מהווה סוף מחרוזת
    listPtr pages;               // מצביע לראש רשימת העמודים שבהם מופיעה המילה שבאינדקס
    struct tstType *left;        // מצביע לתת עץ שמאלי שבו מחרוזות המתחילות בתו הקטן מ-ch
    struct tstType *mid;         // מצביע לתת-עץ אמצעי שבו מחרוזות המתחילות בתו בדיוק ch
    struct tstType *right;       // מצביע לתת-עץ ימני שבו מחרוזות המתחילות בתו הגדול מ-ch
} *indexPtr;
```

שים לב: בצומת המהווה סוף מילה (צומת שבו ערך השדה `isLast` הוא `TRUE`), המשתנה `pages` יכיל מצביע לרשימה מקושרת של העמודים שבהם מופיעה המילה, אחרת – המשתנה `pages` יכיל `NULL`.

להלן איור של עץ אינדקס המילים במבנה המעודכן:



איור ו' לשאלה 1

ניתן לראות כי בצומת המהווה סוף מילה נשמרת רשימה מקושרת של העמודים שבהם מופיעה המילה הזו.

נתונה ספריית פונקציות המכילה, בין היתר, את הפונקציות שבהן נשתמש בסעיפים שלהלן:

כותרת הפונקצייה	תיאור הפונקצייה
<code>void printAllWords(indexPtr node)</code>	<p>הפונקצייה מקבלת את <code>node</code>, שהוא מצביע לצומת כלשהו בעץ אינדקס המילים.</p> <p>הפונקצייה מדפיסה את כל המילים המאוחסנות, החל מהצומת <code>node</code> (כלומר, כל המילים שצומת סוף המילה שלהן נמצא החל מהצומת <code>node</code>), ואת רשימת העמודים שבהם הן מופיעות, כאשר כל מילה ורשימת העמודים שלה יודפסו בשורה נפרדת.</p> <p>לדוגמה: בעבור <code>node</code>, המצביע לראש העץ, הפונקצייה תדפיס את כל המילים בעץ ואת רשימת העמודים שבהם הן מופיעות.</p> <p>כמו כן, המילים יודפסו בסדר מילוני, ומספרי העמודים יודפסו בצורה ממוינת בסדר עולה.</p>
<code>void printWord(indexPtr endNode, boolean withPages)</code>	<p>הפונקצייה מקבלת את <code>endNode</code>, שהוא מצביע לצומת המציין סוף מילה בעץ אינדקס המילים, וכן ערך בוליאני. הפונקצייה מדפיסה את המילה שהאות האחרונה שלה נמצאת בצומת <code>endNode</code>.</p> <p>אם ערכו של <code>withPages</code> הוא <code>TRUE</code>, אזי לאחר שתודפס המילה, תודפס גם רשימת העמודים שבהם היא מופיעה, אחרת – תודפס המילה בלבד (ללא רשימת העמודים).</p>

הנח שהפונקציות האלו כתובות, וניתן להשתמש בהן בכל הסעיפים הבאים בלי לכתוב אותן מחדש. כמו כן, ניתן להשתמש בפונקציות הספרייה `string.h` לטיפול במחרוזות.

ענה על שניים מבין הסעיפים ח'–י' (לכל סעיף – 6 נקודות).

ח. לפניך פונקצייה שכותרתה:

```
void autocomplete(indexPtr root, char* prefix)
```

פונקצייה זו מקבלת את הפרמטרים האלה:


root – מצביע לשורשו של עץ אינדקס המילים של ספר כלשהו

prefix – מחרוזת המייצגת תחילית של מילה (יכולה גם להיות ריקה או להכיל אות יחידה)

הפונקצייה מדפיסה את כל המילים (והעמודים) שבאינדקס המילים, הפותחות בתחילית prefix .

כל מילה ורשימת העמודים שבהם היא מופיעה יודפסו בשורה נפרדת. המילים יודפסו בסדר מילוני, ומספרי העמודים יודפסו בצורה ממוינת בסדר עולה.

הפונקצייה פועלת כמו מנגנון השלמה אוטומטית שמציע מנוע החיפוש של גוגל: מזינים לשורת החיפוש את אותיות התחילית, ומנוע החיפוש מציע רשימה של מילים הפותחות בתחילית זו.



The image shows a search bar with the text 'ba' entered. Below the search bar is a dropdown menu with five suggestions, each preceded by a location pin icon:

- Barcelona Spain
- Bali Indonesia
- Bangkok Thailand
- Banten Indonesia
- Baltimore MD, USA

לדוגמה, זימון הפונקצייה עם הפרמטרים עץ אינדקס המילים המתואר באיור ו', והתחילית "cu", ידפיס את המילים: cup , cups (בשורות נפרדות, כולל רשימת העמודים לכל מילה).

הפונקצייה פועלת כך:

היא עוברת על מחרוזת התחילית prefix , תו אחר תו, תוך כדי מעבר על העץ, כדי לוודא שהתחילית prefix קיימת בעץ במלואה, ובאופן רציף.

אם התחילית קיימת בעץ – הפונקצייה מדפיסה את כל המילים שפותחות בתחילית prefix .

אחרת – הפונקצייה תדפיס את ההודעה "Not Found" ותסתיים.

נוסף על כך, הפונקצייה מטפלת גם במצב שבו המחרוזת prefix ריקה. במקרה זה, הפונקצייה תדפיס את כל המילים שבעץ אינדקס המילים.

בפונקצייה חסרים **חמישה** ביטויים המסומנים במספרים בין סוגריים עגולים. כתוב במחברת הבחינה את מספרי הביטויים החסרים (1)–(5) בסדר עולה, וכתוב ליד כל מספר את הביטוי החסר שהוא מייצג.

```
void autocomplete(indexPtr root, char* prefix)
{
    int pos = 0;
    while (root && pos < _____(1)_____)
    {
        if (prefix[pos] < root->ch)
            root = root->left;
        else
        {
            if (prefix[pos] > root->ch)
                root = root->right;
            else
            {
                if (prefix[pos+1] == '\0' && root->isLast == TRUE)
                {
                    _____(2)_____;
                }
                _____(3)_____;
                pos++;
            }
        }
    }
    if (_____(4)_____)
        printf("Not Found");
    else
        _____(5)_____;
}
```


ט. לפניך פונקציית **רקורסיבית** שכותרתה:

```
int matchPattern(indexPtr root, char* pattern)
```

פונקצייה זו מקבלת את הפרמטרים האלה:

root – מצביע לשורשו של עץ אינדקס המילים של ספר מסוים

pattern – מחרוזת לא ריקה, המייצגת תבנית, המורכבת רק מהתווים הבאים:
אותיות קטנות מתוך האלף-בית האנגלי (a–z) וכן מהתו '?'

הפונקצייה מדפיסה את כל המילים (והעמודים) שבאינדקס המילים, המתאימות לתבנית pattern .

כל מילה ורשימת העמודים שלה יודפסו בשורה נפרדת. המילים יודפסו בסדר מילוני, והעמודים יודפסו בצורה ממוינת בסדר עולה.

כמו כן, הפונקצייה מחזירה את מספר המילים שנמצאו מתאימות לתבנית pattern .

מילה מתאימה לתבנית אם מתקיימים כל התנאים שלהלן:

1. מספר האותיות שבמילה ומספר התווים שבתבנית זהה.
2. כל אות (a–z) המופיעה בתבנית, מופיעה גם במילה באותו המיקום בדיוק.
3. עבור כל תו סימן שאלה (?) המופיע בתבנית, קיימת אות כלשהי במילה, באותו המיקום בדיוק.

דוגמאות:

1. זימון הפונקצייה עם הפרמטרים עץ אינדקס המילים, המתואר באיור ו', והתבנית: "n?" יחזיר את הערך 2 , ויודפסו המילים: many, anna (והעמודים שבהם הן מופיעות), כיוון שהן תואמות את התבנית במדויק.
הסבר: התבנית "n?" מתאימה למילים באורך 4 אותיות **בדיוק** (כמספר **התווים** בתבנית), כאשר האות הראשונה, השנייה, והרביעית שלהן יכולות להיות כל אות שהיא, והאות השלישית שלהן חייבת להיות n .
2. זימון הפונקצייה עם הפרמטרים עץ אינדקס המילים, המתואר באיור ו', והתבנית "m?s" יחזיר את הערך 0 כי לא קיימת מילה באינדקס המילים שמתאימה לתבנית זו.

אופן פעולת הפונקצייה matchPattern דומה לאופן פעולת הפונקצייה search , למעט במצב שבו נתקלים בתו '?' .

במצב זה, במקום לחפש צומת שבו נמצא התו '?' (כי הרי '?' אינו נמצא בצומתי העץ), צריך להמשיך את החיפוש בכל שלושת תתי-העצים כאילו הייתה התאמה.

בפונקציה חסרים **חמישה** ביטויים המסומנים במספרים בין סוגריים עגולים. כתוב במחברת הבחינה את מספרי הביטויים החסרים, (1)–(5) בסדר עולה, וכתוב ליד כל מספר את הביטוי החסר שהוא מייצג.

```
int matchPattern(indexPtr root, char* pattern)
{
    int x1=0, x2=0, x3=0;
    if (!root)
        return 0;
    if (*pattern == '?' || *pattern < root->ch)
        x1 = matchPattern(root->left, pattern);
    if (_____(1)_____)
    {
        if (*(pattern+1) != '\0')
            x2 = _____(2)_____;
        else
            if (root->isLast)
            {
                _____(3)_____;
                _____(4)_____;
            }
    }
    if (*pattern == '?' || *pattern > root->ch)
        x3 = matchPattern(root->right, pattern);
    return _____(5)_____;
}
```

י. לפניך פונקציית **רקורסיבית** שכותרתה:

```
int wordsOnPage(indexPtr root, int pageNum)
```

פונקציית זו מקבלת את הפרמטרים האלה:

root – מצביע לשורשו של עץ אינדקס המילים של ספר כלשהו

pageNum – מספר עמוד בספר

הפונקציית מדפיסה את כל המילים שבאינדקס המילים, הנמצאות בעמוד pageNum .

המילים יודפסו בשורות נפרדות וללא רשימת העמודים שבהם הן מופיעות.

סדר הדפסת המילים יהיה על פי סריקה תחילית (pre-order) של עץ אינדקס המילים.

כמו כן, הפונקציית מחזירה את סך כל המילים הנמצאות בעמוד pageNum .

לדוגמה: זימון הפונקציית wordsOnPage עם הפרמטרים אינדקס המילים, המוצג באיור ו', ועם מספר העמוד 34, ידפיס את המילים: bat , cup , made , ויחזיר את הערך 3 (בהתייחס לאינדקס המילים הנתון).

בפונקציית חסרים **חמישה** ביטויים המסומנים במספרים בין סוגריים עגולים. כתוב במחברת הבחינה את מספרי הביטויים החסרים (1)–(5) בסדר עולה, וכתוב ליד כל מספר את הביטוי החסר שהוא מייצג.

```
int wordsOnPage(indexPtr root, int pageNum)
{
    if (!root)
        return 0;
    int check = 0;
    if(_____(1)_____)
    {
        listPtr p = root->pages;
        while (_____(2)_____)
            p = p->next;
        if (p != NULL && p->pnum == pageNum)
        {
            _____(3)_____;
            printWord(_____(4)_____);
        }
    }
    return _____(5)_____ + wordsOnPage(root->left) +
                                   wordsOnPage(root->mid) +
                                   wordsOnPage(root->right);
}
```

שאלה 2 – שאלת חובה (30 נקודות)

בשאלה זו שישה סעיפים (א'–ו').

ענה על שלושה מבין הסעיפים א'–ד' (לכל סעיף – 8 נקודות),
ועל סעיף אחד נוסף מבין הסעיפים ה'–ו' (לכל סעיף – 6 נקודות).

"יינות שושן" הוא יקב גדול הכולל כמה מרתפי יישון לחביות יין.

הוחלט להקים מערכת ממוחשבת המיועדת לאחסון המידע על מרתפי היין שביקב.

מבנה הנתונים שנבחר הוא רשימה מקושרת, שכל רשומה בה מייצגת **מרתף יין**, וכוללת את השדות שלהלן:

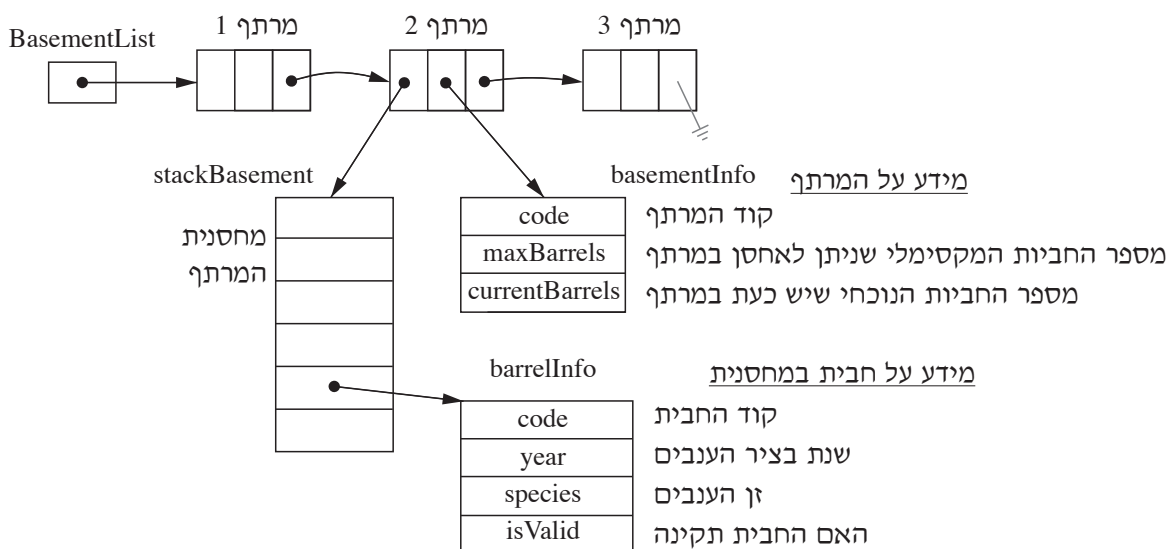
1. מבנה מסוג `basementInfo`, שבו קיים מידע על המרתף (קוד המרתף, מספר החביות המקסימלי שניתן לאחסן במרתף, מספר החביות הנוכחי שיש כעת במרתף);
2. מחסנית חביות היין במרתף;
3. מצביע למרתף הבא.

שנת הבציר קובעת את גיל חבית היין, ועל פיה מסודרות חביות היין במחסנית שבכל מרתף. כלומר, ככל שהחבית ישנה יותר, היא תימצא עמוק יותר במחסנית.

כמו כן, כל איבר במחסנית חביות היין מפנה למבנה `barrelInfo` המייצג טבלה, השומרת את כל המידע הנדרש על חבית היין (קוד החבית, שנת הבציר, זן הענבים וכן אם החבית תקינה).

שים לב: בכל סעיפי השאלה, כאשר נדרש לחשב את סיבוכיות זמן הריצה, אפשר להתייחס ל-N כאל מספר החביות שבמחסנית המרתף, ול-M כאל מספר המרתפים הקיימים ביקב.

באיור שלהלן מתואר מבנה סכמתי חלקי של מבני הנתונים שבהם יאוחסנו נתוני המערכת הממוחשבת:



איור א' לשאלה 2

להלן הגדרות של משתנים גלובליים שיוזכרו בהמשך:

```
#define BARREL_REMOVAL_COST 10
#define BARREL_MOVE_METER 20
typedef enum {FALSE, TRUE} boolean;

typedef struct                // טיפוס המגדיר את טבלת המידע של המרתף
{
    int code;                // קוד המרתף
    int maxBarrels;          // מספר החביות המקסימלי שניתן לאחסן במרתף
    int currentBarrels;      // מספר החביות הנוכחי שיש כעת במרתף
}basementInfo;

typedef struct                // טיפוס המגדיר את טבלת המידע של חבית היין
{
    int code;                // קוד החבית
    int year;                // שנת בציר הענבים
    char species[20];        // זן הענבים
    boolean isValid;         // האם החבית תקינה
}barrelInfo;

typedef struct nodeBasement  // טיפוס המגדיר מרתף ברשימת המרתפים
{
    basementInfo m_basementInfo; // טבלת המידע של המרתף
    STACK basementStack;         // מחסנית חביות היין השייכות למרתף הזה (תפורט בהמשך)
    struct nodeBasement *next;   // מצביע למרתף הבא
}nodeBasementRec, *nodeBasementPtr;

nodeBasementPtr BasementList; // משתנה גלובלי שמצביע לראש רשימת המרתפים
```

מחסנית חביות היין (barrelInfo) היא מבנה מטיפוס STACK המממש מחסנית סטנדרטית.

להלן ארבע פונקציות ספרייה. תוכל להיעזר בהן. אין צורך לממש אותן.

תיאור הפעולה	כותרת הפעולה
הפעולה תדחוף למחסנית s את החבית x	<code>void push(STACK *s, barrelInfo *x);</code>
הפעולה תוציא חבית מראש המחסנית s ותחזיר אותה כערך	<code>barrelInfo pop(STACK *s);</code>
הפעולה תחזיר כערך את החבית שנמצאת בראש המחסנית s מבלי להוציאה מן המחסנית	<code>barrelInfo top(STACK *s);</code>
הפעולה תחזיר TRUE אם המחסנית s ריקה, אחרת - היא תחזיר FALSE	<code>boolean isEmpty(STACK s);</code>

שים לב: סיבוכיות זמן הריצה של כל פעולות המחסנית הנתונות היא $O(1)$.

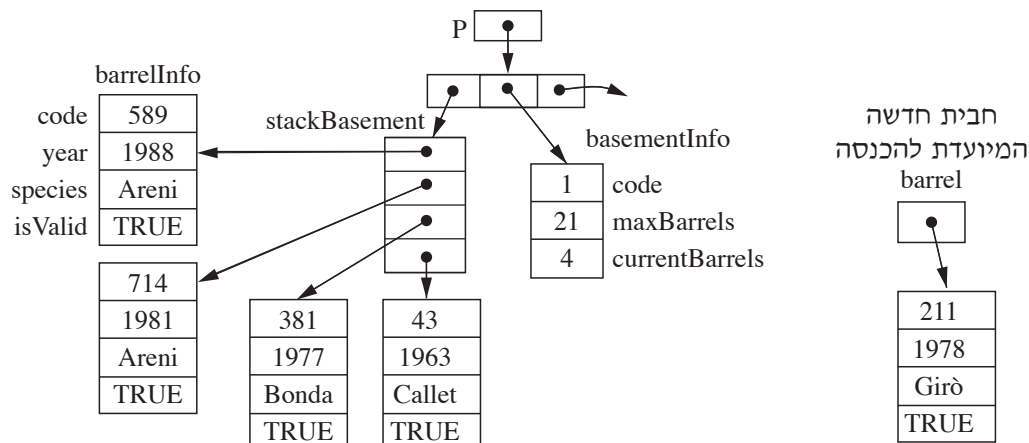
להלן אופן חישוב העלות של הכנסת חבית יין למרתף:

כאשר רוצים לחשב את העלות של הכנסת חבית יין למרתף, יש לחשב את סך החביות שיש להוציא מן המחסנית כדי לשים את החבית הרצויה במיקום הנכון, ואותו להכפיל בעלות הוצאת חבית (BARREL_REMOVAL_COST).

אם מחסנית חביות היין ריקה, אזי עלות ההכנסה היא אפס.

לדוגמה: באיור ב' שלהלן מוצגת חבית חדשה שאותה יש להכניס למרתף שמתואר בחלק השמאלי של האיור.

על-פי האיור, ניתן להכניס את החבית החדשה למרתף, שעליו מצביע p, מכיוון שמספר החביות במרתף כעת הוא 4, ואילו מספר החביות המקסימלי שניתן להכניס אליו הוא 21. כמו כן, ניתן לראות כי כדי להכניס למרתף את החבית החדשה במיקום הנכון מבחינת גיל החבית (1978), יש להוציא מן המרתף שתי חביות (1981 ו-1988). לפיכך, עלות הכנסת החבית תהיה $(2 * \text{BARREL_REMOVAL_COST})$.



איור ב' לשאלה 2

ענה על שלושה מבין הסעיפים א'-ד' (לכל סעיף - 8 נקודות).

א. 1. לפניך הפונקצייה שכותרתה:

```
int extract_cost(nodeBasementPtr basement, barrelInfo *barrel)
```

פונקצייה זו מקבלת את basement, שהוא מצביע למרתף מסוים, ואת החבית barrel.

הפונקצייה מחשבת ומחזירה את העלות הכרוכה בהכנסת החבית barrel למרתף שעליו מצביע basement.

הערה: הפונקצייה מחשבת את העלות הכרוכה בהכנסת החבית, אך אינה מוסיפה את החבית למרתף.

בפונקצייה חסרים **ארבעה** ביטויים המסומנים במספרים בין סוגריים עגולים. כתוב במחברת הבחינה את מספרי הביטויים החסרים (1)–(4) בסדר עולה, וכתוב ליד כל מספר את הביטוי החסר שהוא מייצג.

```
int extract_cost(nodeBasementPtr basement, barrelInfo *barrel)
{
    if (_____ (1) _____)
        return 0;
    int count = 0;
    STACK tmp = NULL;
    while (!isEmpty(basement->basementStack) && _____ (2) _____ > barrel->year)
    {
        barrelInfo x = _____ (3) _____;
        push(&tmp, &(x));
        count++;
    }
    while (_____ (4) _____)
    {
        barrelInfo x = pop(&(tmp));
        push(&(basement->basementStack), &x);
    }
    return count * BARREL_REMOVAL_COST;
}
```

2. מהי סיבוכיות זמן הריצה של הפונקצייה extract_cost ?

ב. 1. לפיך פונקציה **רקורסיבית** שכותרתה:

```
boolean addBarrel(nodeBasementPtr basement, barrelInfo *barrel)
```

פונקציה זו מקבלת את basement, שהוא מצביע למרתף, ואת החבית barrel, ומוסיפה אותה למחסנית החביות של המרתף במקום הנדרש. כמו כן, הפונקציה מעדכנת את מספר החביות הנוכחי שיש במרתף. הפונקציה מחזירה TRUE אם החבית הוכנסה בהצלחה, אחרת - הפונקציה מחזירה FALSE.

בפונקציה חסרים **חמישה** ביטויים המסומנים במספרים בין סוגריים עגולים. כתוב במחברת הבחינה את מספרי הביטויים החסרים (1)-(5) בסדר עולה, וכתוב ליד כל מספר את הביטוי החסר שהוא מייצג.

```
boolean addBarrel(nodeBasementPtr basement, barrelInfo *barrel)
{
    if (isEmpty(basement->basementStack))
    {
        push(&(basement->basementStack), barrel);
        basement->m_basementInfo.currentBarrels++;
    }
    else if (basement->m_basementInfo.currentBarrels == _____(1)_____)
    {
        return FALSE;
    }
    else if (_____(2)_____ > barrel->year)
    {
        barrelInfo x = pop(_____(3)_____);
        _____(4)_____;
        push(&(basement->basementStack), &x);
    }
    else
    {
        push(&(basement->basementStack), barrel);
        _____(5)_____;
    }
    return TRUE;
}
```

2. מהי סיבוכיות זמן הריצה של הפונקציה addBarrel ?

ג. 1. לפיך פונקציית שכותרתה:

```
nodeBasementPtr getBestBasement(barrelInfo* barrel)
```

פונקציית זו מקבלת את barrel, שהוא מצביע לחבית שיש להכניס ליקב.

הפונקציית עוברת על רשימת המרתפים, ומחזירה מצביע לאחד מבין כל המרתפים שאינם מלאים, אשר עלות ההכנסה אליו היא המינימלית. אם כל המרתפים מלאים, הפונקציית מחזירה את הערך NULL.

הערות:

I. הפונקציית יכולה להשתמש בפונקציות שמומשו בסעיפים הקודמים.

II. הפונקציית מחזירה מצביע למרתף המבוקש, אם ישנו כזה, אך אינה מטפלת בהכנסת החבית למרתף הזה.

בפונקציית חסרים **חמישה** ביטויים המסומנים במספרים בין סוגריים עגולים. כתוב במחברת הבחינה את מספרי הביטויים החסרים (1)-(5) בסדר עולה, וכתוב ליד כל מספר את הביטוי החסר שהוא מייצג.

```
nodeBasementPtr getBestBasement(barrelInfo* barrel)
{
    nodeBasementPtr p = BasementList;
    nodeBasementPtr result = p;
    int best = extract_cost(result, barrel);
    while (p != NULL && _____(1)_____ == p->m_basementInfo.maxBarrels)
    {
        p = p->next;
    }
    if (_____(2)_____)
        return NULL;
    result = p;
    best = extract_cost(result, barrel);
    p = p->next;
    while (p != NULL)
    {
        int x = _____(3)_____;
        if (x < best && p->m_basementInfo.currentBarrels < p->m_basementInfo.maxBarrels)
        {
            best = x;
            _____(4)_____ = p;
        }
        _____(5)_____;
    }
    return result;
}
```

2. מהי סיבוכיות זמן הריצה של הפונקציית getBestBasement ?

ד. 1. לפניך פונקציית שכותרתה:

```
void insertBarrelToWinery(barrelInfo *barrel)
```

הפונקצייה מקבלת חבית barrel , ואמורה להוסיף אותה למרתף שעלות ההוספה אליו היא המינימלית.

אם החבית הוכנסה בהצלחה - הפונקצייה תדפיס OK .

אם לא ניתן להוסיף את החבית לשום מרתף ביקב, הפונקצייה תדפיס Error .

הפונקצייה משתמשת בחלק מן הפעולות שמומשו בסעיפים הקודמים.

בפונקצייה חסרים שני ביטויים המסומנים במספרים בין סוגריים עגולים. כתוב במחברת הבחינה את מספרי הביטויים החסרים (1)–(2) בסדר עולה, וכתוב ליד כל מספר את הביטוי החסר שהוא מייצג.

```
void insertBarrelToWinery(barrelInfo *barrel)
{
    nodeBasementPtr p = _____(1)_____;
    if (p != NULL && _____(2)_____)
    {
        printf("OK\n");
    }
    else
    {
        printf("Error\n");
    }
}
```

2. מה סיבוכיות זמן הריצה של הפונקצייה insertBarrelToWinery ?

כחלק מן התחזוקה השוטפת של היקב צריך להעביר חביות ממרתף אחד למרתף אחר.

הנח כי החביות שמעוניינים להעביר נמצאות בראש המחסנית.

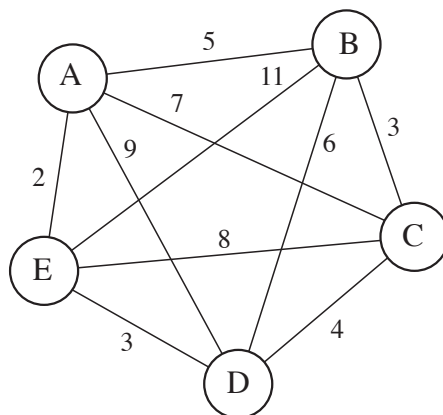
עלות ההעברה של חבית ממרתף למרתף מושפעת מכמה גורמים:

1. המרחק בין מרתף המקור ובין מרתף היעד;

2. מספר החביות שנדרש להוציא ממרתף היעד כדי להכניס את החבית שבראש מרתף המקור למקומה המתאים במרתף היעד;

3. מקום פנוי במרתף היעד להכנסת חבית יין.

בדוגמה שלהלן נתונה רשת המתארת את המרחקים בין המרתפים הנמצאים ביקב כלשהו.
ביקב הזה ישנם חמישה מרתפים (A,B,C,D,E), והקשתות מציגות את המרחקים בין כל זוג מרתפים.
הנח כי ברשת יש קשת ישירה בין כל זוג מרתפים.



איור ג' לשאלה 2

במקום לייצג את המרחקים שבין המרתפים באמצעות רשת, הוחלט לעשות זאת על-ידי רשימות מקושרות.
לכל מרתף הוסיפו מצביע לרשימה מקושרת שכל איבר בה מוגדר כמתואר להלן:

```
typedef struct proximityNode
{
    int distance; // המרחק אל המרתף האחר (בהתאם לרשת)
    struct nodeBasementPtr *p_basement; // מצביע למרתף האחר (בהתאם לרשת)
    struct proximityNode *next; // מצביע לאיבר הבא ברשימה (מרחק)
}proximityNodeRec, *proximityNodePtr;
```

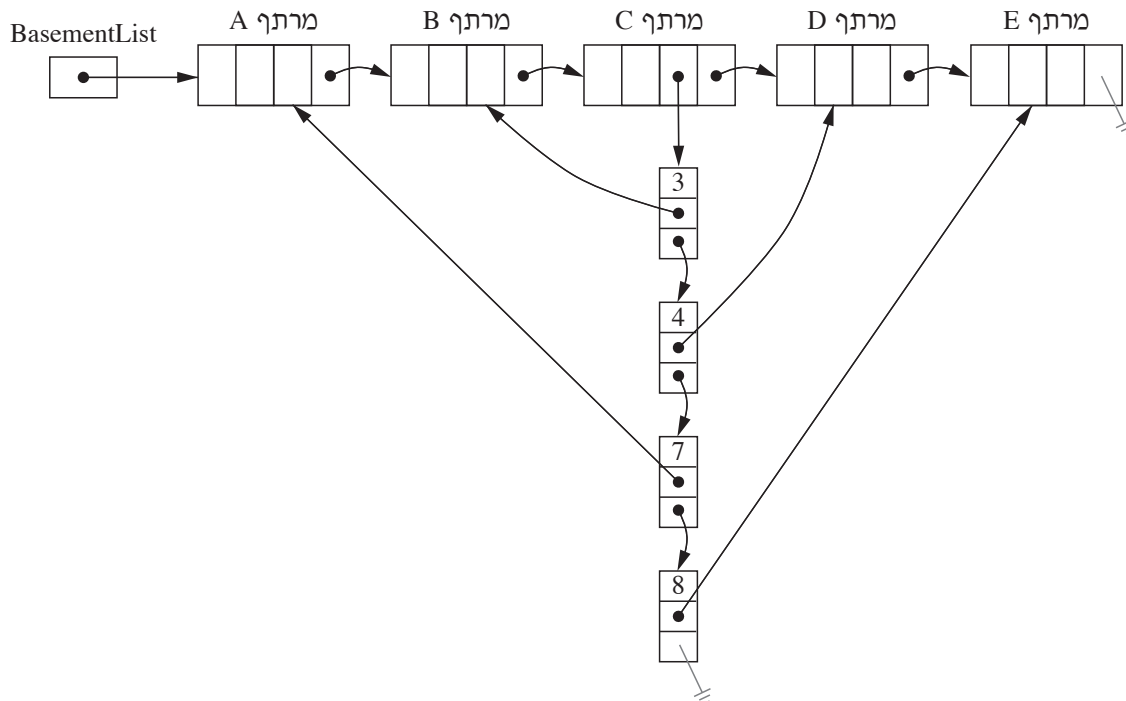
כמו כן עודכנה גם רשימת המרתף כמתואר להלן:

```
typedef struct nodeBasement
{
    basementInfo m_basementInfo;
    struct proximityNode *_basementProximity;
    STACK basementStack;
    struct nodeBasement *next;
}nodeBasementRec, *nodeBasementPtr;
```

השדה שנוסף הוא מצביע לרשימה מקושרת מטיפוס proximityNode, הממוינת על-פי השדה distance בסדר עולה.

באיור שלהלן ניתן לראות את השינוי שחל במבנה:

בעבור מרתף C מתוארת הרשימה המקושרת של המרחקים, אשר עליה מצביע _basementProximity מתוך רשימת המרתף. כמו כן, ניתן לראות כי כל אחד מן הצמתים שברשימת המרחקים מצביע בהתאמה למרתף אחר ברשימת המרתפים.



איור ד' לשאלה 2

תזכורת:

עלות העברת חבית ממרתף אחד למרתף אחר שווה לסכום של שתי העלויות שלהלן:

1. עלות הכנסת חבית למרתף היעד: $(\text{BARREL_REMOVAL_COST}) * \text{מספר החביות שנדרש להוציא ממרתף היעד}$.

2. עלות המרחק: $(\text{BARREL_MOVE_METER}) * \text{המרחק אל מרתף היעד}$.

לצורך פתרון הסעיפים הבאים נתונות הפונקציות הבאות:

התיאור	כותרת הפונקצייה
הפונקצייה מקבלת את <code>basement</code> , שהוא מצביע למרתף. היא מוציאה חבית מראש מחסנית החביות של המרתף שעליו מצביע <code>basement</code> , מעדכנת את שדה <code>currentBarrels</code> שלו בהתאמה, ומחזירה כערך את החבית שהוצאה.	<code>barrelInfo removeBarrel(nodeBasementPtr basement);</code>
הפונקצייה מקבלת את <code>basement</code> , שהוא מצביע למרתף. היא מוציאה ממנו את כל החביות הפגומות, ומעדכנת את שדה <code>currentBarrels</code> של המרתף. הערה: בעבור חבית פגומה, ערכו של השדה <code>isValid</code> יהיה <code>FALSE</code> .	<code>void removeBadBarrels(nodeBasementPtr basement);</code>
הפונקצייה מקבלת את <code>fromB</code> , שהוא מצביע למרתף המקור, את <code>toB</code> שהוא מצביע למרתף היעד, ואת החבית <code>barrel</code> שנמצאת בראש המחסנית ומיועדת להעברה. הפונקצייה מחשבת ומחזירה את העלות הכוללת של העברת החבית <code>barrel</code> ממרתף המקור למרתף היעד.	<code>int costWithDist(nodeBasementPtr fromB, nodeBasementPtr toB, barrelInfo *barrel);</code>

ענה על אחד מבין הסעיפים ה'–ו' (לכל סעיף – 6 נקודות).

ה. לפניך פונקצייה שכותרתה:

```
nodeBasementPtr findBasementToMove (nodeBasementPtr current)
```

פונקצייה זו מקבלת מצביע למרתף מסוים (`current`) ומחזירה מצביע למרתף אחר שמקיים את התנאים שלהלן:

1. עלות העברת החבית, שנמצאת בראש המחסנית שבמרתף `current`, אל המרתף הזה היא הקטנה ביותר ביחס למרתפי היקב האחרים.

2. במרתף הזה יש מקום פנוי עבור חבית נוספת.

אם לא נמצא מרתף העונה על הדרישות, הפונקצייה תחזיר את הערך `NULL`.

שים לב: פונקצייה זו מחזירה מצביע למרתף המקיים את התנאים הללו, אם ישנו כזה, אך אינה מטפלת בהעברת החבית למרתף הזה.

בפונקצייה חסרים **חמישה** ביטויים המסומנים במספרים בין סוגריים עגולים. כתוב במחברת הבחינה את מספרי הביטויים החסרים (1)–(5) בסדר עולה, וכתוב ליד כל מספר את הביטוי החסר שהוא מייצג.

```
nodeBasementPtr findBasementToMove (nodeBasementPtr current)
{
    proximityNodePtr p_proximity = current->_basementProximity;
    int x1,x2;
    barrelInfo b = top(&(current->basementStack));
    nodeBasementPtr p = p_proximity->p_basement;
    nodeBasementPtr result = NULL;
    while (p_proximity != NULL && p->m_basementInfo.currentBarrels == p->m_basementInfo.maxBarrels)
    {
        p_proximity = p_proximity->next;
        if (p_proximity != NULL)
            p = _____(1)_____;
        else
            return NULL;
    }
    x2 = _____(2)_____;
    result = p;
    while (p_proximity->next != NULL)
    {
        p_proximity = _____(3)_____;
        p = _____(4)_____->p_basement;
        if (p->m_basementInfo.currentBarrels < p->m_basementInfo.maxBarrels)
        {
            x1 = costWithDist(current, p, &b);
            if (_____(5)_____)
            {
                x2 = x1;
                result = p;
            }
        }
    }
    return result;
}
```

1. באחד המרתפים התגלתה רטיבות, והוחלט לפנות ממנו את כל חביות היין, ולהעביר את החביות שלא נפגמו למרתפים פנויים אחרים.

לפניך פונקצייה שכותרתה:

```
int *emptyBasement (nodeBasementPtr basement)
```

פונקצייה זו מקבלת את basement, שהוא מצביע למרתף שאותו רוצים לפנות.

הפונקצייה מסירה את החביות הפגומות, ומטפלת בהעברת החביות שאינן פגומות למרתפים שעלות ההעברה אליהם היא המינימלית, אם יש מקום פנוי עבורן במרתפי היעד.

הפונקצייה מחזירה מערך מטיפוס int בעל שלושה תאים כדלהלן:

1. בתא הראשון ישוכן מספר החביות התקינות שהוצאו מן המרתף.
2. בתא השני ישוכן העלות הכוללת של העברת החביות.
3. בתא השלישי ישוכן מספר החביות שנותרו במרתף, אם לא היה מקום פנוי עבורן במרתפים האחרים.

בפונקצייה חסרים **חמישה** ביטויים המסומנים במספרים בין סוגריים עגולים. כתוב במחברת הבחינה את מספרי הביטויים החסרים (1)–(5) בסדר עולה, וכתוב ליד כל מספר את הביטוי החסר שהוא מייצג.

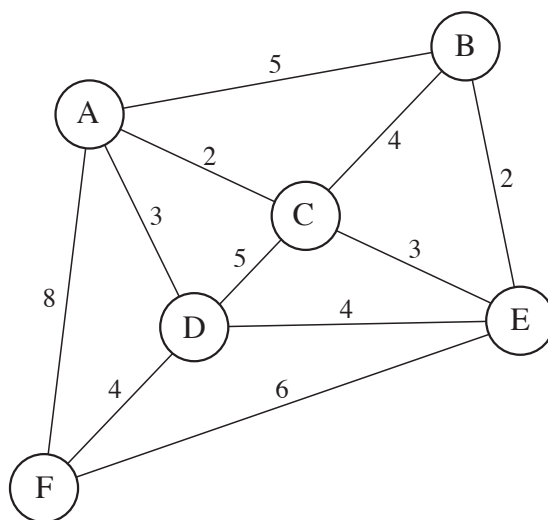
```
int *emptyBasement(nodeBasementPtr basement)
{
    int result[3] = {0,0,0};
    removeBadBarrels(basement);
    int x1 = 0, cost = 0, count = 0, total = 0;
    nodeBasementPtr p = BasementList;
    while (p != NULL)
    {
        if (p != basement)
        {
            total += p->m_basementInfo.maxBarrels - p->m_basementInfo.currentBarrels;
        }
        p = p->next;
    }
    while (basement->m_basementInfo.currentBarrels > 0 && total > 0)
    {
        p = _____(1)_____;
        barrelInfo b = _____(2)_____;
        cost = _____(3)_____ (basement, p, &b);
        if (addBarrel(p, &b))
        {
            x1 += cost;
            count++;
            total--;
        }
    }
    result[0] = count;
    _____(4)_____;
    result[2] = _____(5)_____;
    return result;
}
```


פרק שני (30 נקודות)

בפרק זה יש שתי שאלות – 3-4 ובהן יחד 16 סעיפים.
ענה על שמונה סעיפים בסך-הכול משתי השאלות יחד (לכל סעיף – 3.75 נקודות).

שאלה 3

בשאלה זו 8 סעיפים (לכל סעיף – 3.75 נקודות).
בכל סעיף נתונות ארבע תשובות, שרק אחת מהן נכונה. בכל סעיף שבחרת לענות עליו, בחר את התשובה הנכונה, והקף בעיגול את הספרה המייצגת אותה בדף התשובות שבנספח א'.
א. נתון הגרף שלהלן:



איור א' לשאלה 3

לאחר הרצת אלגוריתם דייקסטרה על הגרף החל מהצומת A יתקבל עץ המסלולים הקצרים ביותר.
מאילו קשתות יהיה מורכב העץ הזה?

1. AC, AD, AB, CE, DF
2. AC, BE, CE, AD, DF
3. AC, AD, CE, CB, DF
4. AC, AD, CE, CB, AF

ב. בחר בהיגד הנכון מבין ההיגדים שלהלן:

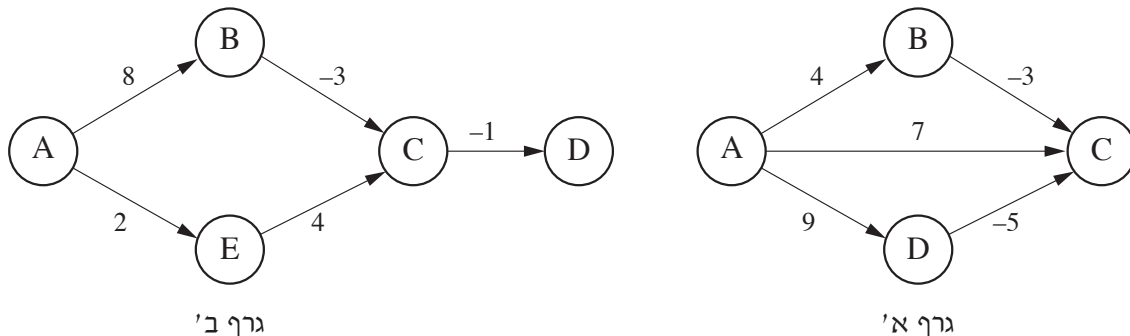
1. לכל גרף יש עץ פורש מינימלי יחיד
2. מטריצת הסמיכויות של גרף מכוון ושל גרף לא מכוון בעלי אותם צמתים ואותן קשתות תהיה זהה תמיד
3. זמן הריצה הנדרש למציאת מסלול קצר ביותר בין שני צמתים בגרף לא ממושקל זהה לזמן הריצה הנדרש למציאת מסלולים קצרים מצומת המקור (צומת כלשהו בגרף) לכל יתר הצמתים בגרף
4. אם משקלי הקשתות בגרף שונים זה מזה, אזי המסלול הקצר ביותר בין שני צמתים בגרף הוא יחיד

ג. בחר בהיגד הנכון מבין ההיגדים שלהלן:

1. הוספת קבוע n שלם וחיובי לכל משקלי הקשתות בגרף לא תשפיע על חישוב המסלול הקצר ביותר בין שני צמתים בגרף
2. הוספת קבוע n שלם וחיובי לכל משקלי הקשתות בגרף לא תשנה את מבנה העץ הפורש המינימלי של הגרף
3. אם בגרף ממושקל שאינו מכוון יש קשת שמשקלה גדול ממשקל כל אחת מיתר הקשתות שבגרף, אזי קשת זו אינה יכולה להיות בעץ הפורש המינימלי של הגרף
4. לאחר הרצה של האלגוריתמים BFS ו-KRUSKAL על גרף יתקבלו תמיד שני עצים זהים

ד. להלן שני גרפים מכוונים.

נתון כי מריצים את אלגוריתם דייקסטרה על הגרפים, החל מצומת A שבכל גרף.



איור ב' לשאלה 3

בחר בהיגד הנכון מבין ההיגדים שלהלן:

1. אלגוריתם דייקסטרה יחזיר בעבור גרף א' את המסלול הקצר ביותר מ-A לכל יתר הצמתים בגרף
2. אלגוריתם דייקסטרה יחזיר בעבור גרף ב' את המסלול הקצר ביותר מ-A לכל יתר הצמתים בגרף
3. ישנו צומת אחד בגרף א' שבעבורו אלגוריתם דייקסטרה יחזיר מסלול שגוי
4. עבור כל הצמתים בגרף ב' אלגוריתם דייקסטרה יחזיר מסלול שגוי

ה. להלן כמה שלבי מיון של מערך מספרים. שלבי המיון אינם בהכרח עוקבים, אך בכל שורה מתואר שלב מיון מתקדם יותר ביחס לשורה הקודמת.

17, 8, 10, 5, 13, 1, 7, 25, 20

8, 10, 5, 13, 1, 7, 17, 25, 20

5, 1, 7, 8, 10, 13, 17, 20, 25

מהי סיבוכיות זמן הריצה של אלגוריתם המיון אשר על פיו מיון המערך? התייחס למקרה הגרוע ביותר.

1. $O(\log n)$

2. $O(n^2 \log n)$

3. $O(n)$

4. $O(n^2)$

ו. $T(n) = 49T\left(\frac{n}{7}\right) + n \log n$ היא פונקציית זמן הריצה של אלגוריתם מסוים, הפועל על קלט שגודלו n .

מהי סיבוכיות זמן הריצה של האלגוריתם?

1. $\Theta(n^2)$

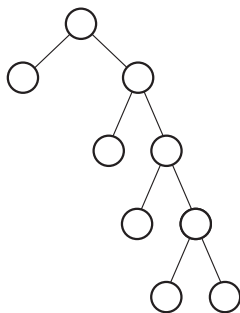
2. $\Theta(n^2 \log n)$

3. $\Theta(n \log n)$

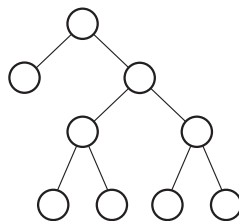
4. $\Theta(n \log^2 n)$

ז. נתון אלף-בית שבו 5 אותיות, ונתון כי השכיחויות של כל האותיות שוות.

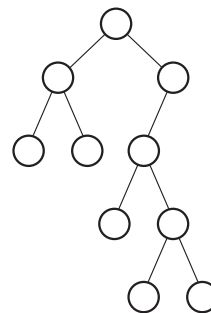
איזה מבין העצים שלהלן הוא עץ הופמן המתאים לאלף-בית הזה?



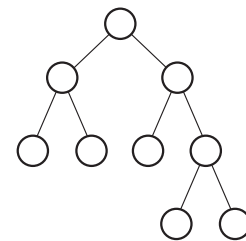
עץ ד'



עץ ג'



עץ ב'



עץ א'

1. עץ א'

2. עץ ב'

3. עץ ג'

4. עץ ד'

ח. נתונים שני עצי חיפוש בינאריים T_1 ו- T_2 , שבכל אחד מהם יש n צמתים, וכל צומת מכיל ערך מספרי. נתונות שתי פונקציות בפסאודו-קוד: $ProcessTree$ ו- $CheckTrees$, כאשר הפונקצייה $CheckTrees$ משתמשת בפונקצייה $ProcessTree$, והעברת הפרמטרים לפונקצייה $ProcessTree$ היא by reference. הנח כי ניתן להשתמש בשני מערכים A_1 ו- A_2 בגודל n , העשויים להכיל ערכים מספריים, וכן ש- ind הוא משתנה גלובלי.

ProcessTree(T, A)

if ($T \neq \text{NULL}$) then

$ProcessTree(T.\text{left}, A)$

$A[ind] \leftarrow T.\text{value}$

$ind \leftarrow ind + 1$

$ProcessTree(T.\text{right}, A)$

CheckTrees(T_1, T_2, n)

$ind \leftarrow 0$

$ProcessTree(T_1, A_1)$

$ind \leftarrow 0$

$ProcessTree(T_2, A_2)$

$i \leftarrow 0$

$flag \leftarrow \text{true}$

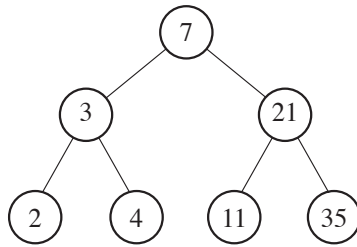
while ($i \leq n-1$) and $flag$ do

$flag \leftarrow A_1[i] == A_2[i]$

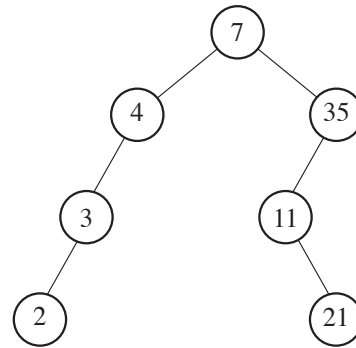
$i \leftarrow i + 1$

return $flag$

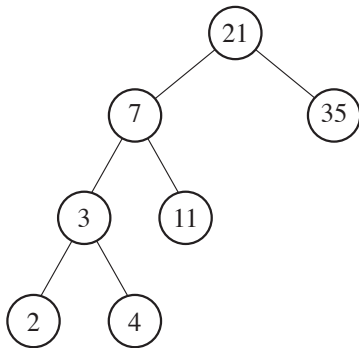
לפניך ארבעה עצים המסומנים באותיות D-A.



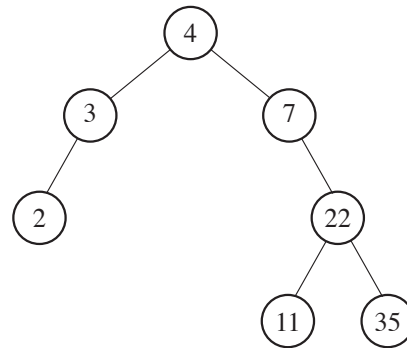
B



A



D



C

איור ד' לשאלה 3

בחר בהיגד שאינו נכון מבין ההיגדים שלהלן:

1. בעבור הקריאה `CheckTrees(A, B, 7)` יוחזר הערך `TRUE`.
2. בעבור הקריאה `CheckTrees(A, C, 7)` יוחזר הערך `FALSE`.
3. בעבור הקריאה `CheckTrees(A, D, 7)` יוחזר הערך `TRUE`.
4. בעבור הקריאה `CheckTrees(D, B, 7)` יוחזר הערך `FALSE`.

שאלה 4

בשאלה זו 8 סעיפים (לכל סעיף – 3.75 נקודות).

בכל סעיף נתונות ארבע תשובות, שרק אחת מהן נכונה. בכל סעיף שבחרת לענות עליו, בחר את התשובה הנכונה, והקף בעיגול את הספרה המייצגת אותה, בדף התשובות שבנספח א'.

א. הוצאת ספרים כלשהי מעוניינת לנהל מעקב אחר היקף המכירות של n כותרים שונים בפרק זמן מסוים. כל כותר מיוצג על-ידי ברקוד.

ברצוננו לתכנן מבנה נתונים שיאפשר אחסון ותחזוקת מידע על מכירת הכותרים של הוצאת הספרים. מבנה הנתונים הזה צריך לממש כל אחת מן הפעולות הבאות בסיבוכיות הזמן הנדרשת:

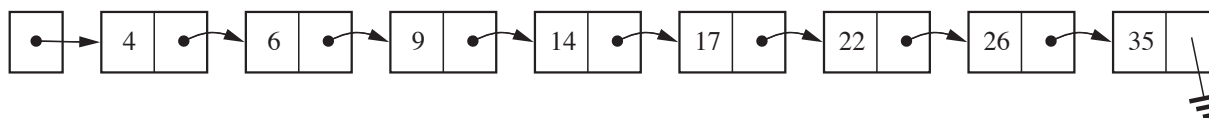
הפעולה	תיאור הפעולה	סיבוכיות נדרשת
INSERT(k, S)	הכנסת כותר שכבר נמכר ונושא את הברקוד k לתוך מבנה הנתונים S .	$O(\log n)$
SEARCH(k, S)	חיפוש במבנה הנתונים S אחר כותר הנושא את הברקוד k .	$O(\log n)$
MOST-POPULAR(S)	החזרת הברקוד של הכותר שנמכרו ממנו מספר העותקים הרב ביותר מתוך מבנה הנתונים S .	$O(1)$

בחר מבין האפשרויות שלהלן את מבנה הנתונים המתאים למימוש פעולות אלה:

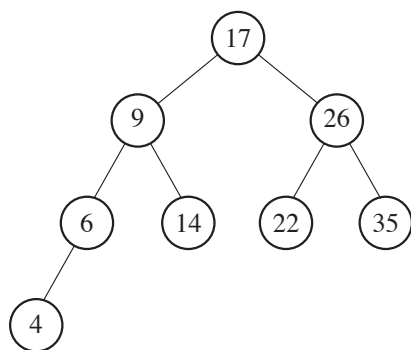
1. מבנה הנתונים יהיה מורכב מעץ AVL בלבד. המפתח של כל איבר בו יהיה ברקוד הכותר, ונוסף על כך תישמר בכל איבר שכיחות רכישת הכותר
2. מבנה הנתונים יהיה מורכב מעץ חיפוש בינארי, שמפתח כל איבר בו יהיה ברקוד הכותר, וכן מערימה שמפתח כל איבר בה הוא שכיחות רכישת הכותר. מכל איבר בעץ יהיה מצביע לאיבר המתאים בערימה
3. מבנה הנתונים יהיה מורכב מעץ AVL, שמפתח כל איבר בו יהיה ברקוד הכותר, וכן מערימה שמפתח כל איבר בה הוא שכיחות רכישת הכותר. מכל איבר בערימה יהיה מצביע לאיבר המתאים בעץ
4. מבנה הנתונים יהיה מורכב מעץ AVL בלבד. המפתח של כל איבר בו יהיה ברקוד הכותר, ונוסף על הרשומה של כל איבר תשורשר רשימת העותקים (מופעים) שנמכרו מאותו הכותר

- ב. נתונה רשימת מספרים מקושרת הממוינת בסדר עולה. ניתן לבנות מרשימה זו עץ בינארי השקול לה, כך:
לשורש העץ נבחר את האיבר הנמצא במקום $\left\lfloor \frac{n}{2} + 1 \right\rfloor$ ברשימה, כאשר האיבר הראשון ברשימה נמצא במקום $n=1$.
איבר זה מחלק את הרשימה לשני חלקים. תת-העץ הימני ייבנה באופן זהה מאיברי החלק הראשון, ותת-העץ השמאלי ייבנה באופן דומה מאיברי החלק השני.

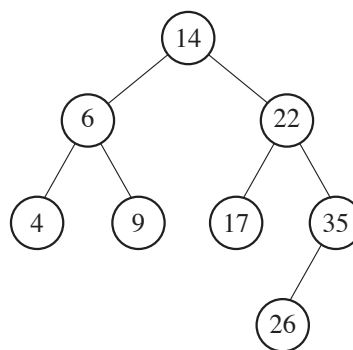
להלן הרשימה המקושרת:



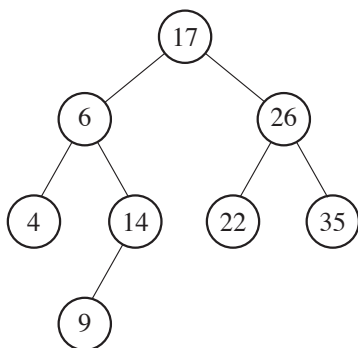
מהו העץ המתקבל בהפעלת האלגוריתם שתואר לעיל?



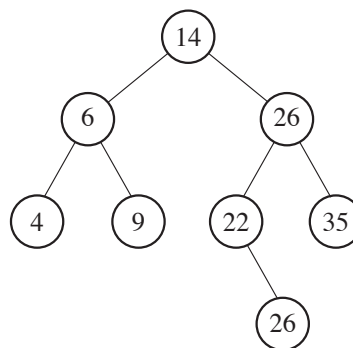
2



1



4



3

ג. לפניך היגדים העוסקים בחיפוש מספר בעץ חיפוש בינארי לעומת חיפוש מספר ברשימה מקושרת הממוינת בסדר עולה. בחר בהיגד הנכון מבין ההיגדים שלהלן.

1. חיפוש מספר כלשהו יהיה תמיד יעיל יותר ברשימה מקושרת הממוינת בסדר עולה לעומת עץ חיפוש בינארי
2. חיפוש המספר הקטן ביותר יהיה יעיל יותר ברשימה מקושרת הממוינת בסדר עולה לעומת עץ חיפוש בינארי
3. חיפוש מספר כלשהו יהיה תמיד יעיל יותר בעץ חיפוש בינארי לעומת רשימה מקושרת הממוינת בסדר עולה
4. חיפוש מספר בעץ חיפוש בינארי וחיפוש מספר ברשימה מקושרת הממוינת בסדר עולה יעילים באותה המידה

ד. להלן מטריצת סמיכויות המתארת את הגרף G :

$$\begin{pmatrix} 0 & 1 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \end{pmatrix}$$

בחר בהיגד הנכון מבין ההיגדים שלהלן:

1. בגרף G יש שני רכיבי קשירות
2. בגרף G יש שני רכיבי קשירות חזקה
3. בגרף G יש שלושה רכיבי קשירות
4. בגרף G יש שלושה רכיבי קשירות חזקה

ה. בחברת המונה 2,500 עובדים מעוניינים לשמור את נתוני העובדים בטבלת גיבוב (hash table) שהמפתח שלה הוא מספר הזהות של העובד.

בחר בהיגד הנכון מבין ההיגדים שלהלן:

1. יש לשמור את נתוני העובדים בטבלת גיבוב סגורה כדי למנוע בעיות התנגשות
2. יש לשמור את נתוני העובדים בטבלת גיבוב פתוחה כיוון שלא צפויות התנגשויות מיעון
3. יש לשמור את נתוני העובדים בטבלת גיבוב סגורה כדי למנוע ניסיונות מיעון של פונקציית הגיבוב
4. יש לשמור את נתוני העובדים בטבלת גיבוב פתוחה כדי למנוע את עקרון שובך היונים ואת הגדלת סיבוכיות זמן הריצה

1. נתון קטע קוד, הפועל על קלט שגודלו n :

```
for (i = 1; i < n; i++)
{
    for (j = 1; j < i; j++)
    {
        for (z = j; z <= i; z++)
        {
            for (k = z; k < i; k++)
            {
                printf("hello");
            }
        }
    }
}
```

מהי סיבוכיות זמן הריצה של קטע הקוד?

1. $\Theta(n^3 \log n)$

2. $\Theta(n^4)$

3. $\Theta(n^2 \log^2 n)$

4. $\Theta(n^3)$

2. להלן כמה שלבי מיון של מערך מספרים. שלבי המיון אינם בהכרח עוקבים, אך בכל שורה מתואר שלב מיון מתקדם יותר ביחס לשורה הקודמת.

15 , 30 , 9 , 7 , 40 , 16 , 2 , 20

40 , 30 , 16 , 20 , 15 , 9 , 2 , 7

30 , 20 , 16 , 7 , 15 , 9 , 2 , 40

מהי סיבוכיות זמן הריצה של אלגוריתם המיון אשר על פיו מיון המערך? התייחס למקרה הגרוע ביותר.

1. $O(\log n)$

2. $O(n \log n)$

3. $O(n)$

4. $O(n^2)$

ח. בסעיף זה נדרש לקודד הודעה באמצעות קוד הופמן.

נתון האלף-בית $\Sigma = \{a, b, c, d, e\}$, ונתונה גם טבלת השכיחויות של האותיות:

a	b	c	d	e
16	8	4	2	1

a b a a b a c a b a a b a c d a b a a b a c a b a a b a c d e

כמה ביטים תכיל ההודעה הבאה לאחר שנקודד אותה?

1. 65 ביטים
2. 61 ביטים
3. 56 ביטים
4. 53 ביטים

בהצלחה!

נספח א': דף תשובות לפרק השני – שאלות 3 ו-4,

לשאלון 714911, אביב תשפ"א

מקור מידע נבחן

הדבק את מדבקת הנבחן שלך במקום המיועד לכך, והדק את הדף הזה למחברת הבחינה שלך.

הקף בעיגול את הספרה המייצגת את התשובה הנכונה לכל סעיף.

שאלה 4					שאלה 3				
4	3	2	1	סעיף א	4	3	2	1	סעיף א
4	3	2	1	סעיף ב	4	3	2	1	סעיף ב
4	3	2	1	סעיף ג	4	3	2	1	סעיף ג
4	3	2	1	סעיף ד	4	3	2	1	סעיף ד
4	3	2	1	סעיף ה	4	3	2	1	סעיף ה
4	3	2	1	סעיף ו	4	3	2	1	סעיף ו
4	3	2	1	סעיף ז	4	3	2	1	סעיף ז
4	3	2	1	סעיף ח	4	3	2	1	סעיף ח

המונח	תרגום המונח		
	ערבית	רוסית	אנגלית
אחזור	استرجاع	Возврат, извлечение	retrieval
איבר	مُتَغَيِّر / عضو	Элемент	item
אקראי	عشوائي	Случайный	random
אתחול	قيمة بدائية	Инициализация	initialization
זמן ריצה	مدّة التنفيذ	Время работы	run time
טבלת ערבול (גיבוב)	جدول الخلط	Хеш-таблица	hash table
טיפוס	نوع	Тип	type
מחסנית	باغة	Стек	stack
מטריצת סמיכויות	جدول الحدود الزميلة	Матрица смежности	adjacency matrix
מיון טופולוגי	تصنيف	Топологическая сортировка	topological sorting
מסלול	مسار	Путь	path
מערך דינמי	مصفوفة غير ثابتة	Динамический массив	dynamic array
מצביע	مُؤَشِّر	Указатель	pointer
משתנה גלובלי	مُتَغَيِّر عام	Глобальная переменная	global variable
סדרה	سلسلة	Последовательность	series
סיבוכיות	تعقيد	Сложность (вычислений)	complexity
עדיפות	أولوية	Приоритет	preference
עץ חיפוש בינארי מאוזן	شجرة بحث ثنائي متوازنة	сбалансированное двоичное дерево поиска	balanced binary search tree
ערך מוחלט	قيمة مُطلَقة	модуль	absolute value
ערימה	كومة	Куча	heap
ערימה בינארית	كومة ثنائية	Двоичная куча	binary heap
פונקצייה רקורסיבית	دالة أو عملية تراجعية	Рекурсивная функция	recursive function
צומת	مُفْتَرَق	Узел	node

תרגום המונח			המונח
אנגלית	רוסית	ערבית	
vertex	вершина	رأس	קודקוד
arc	Дуга	وصلة	קשת
record	Запись (элемент структуры данных)	سَجَل	רשומה
linking field	Поле, содержащее ссылку	حقل رابط	שדה קישור
root	Корень	جذر	שורש
sub-tree	Поддерево	شجرة فرعية	תת־עץ