מדינת ישראל
משרד החינוך

סוג הבחינה: גמר לבתי־ספר לטכנאים ולהנדסאים
מועד הבחינה: אביב תש"ף, 2020
סמל השאלון: 714911
נספחים: א. דף תשובות לשאלות 3 ו־4
ב. מילון מונחים

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:

1. את התשובות לשאלות 1 ו־2 יש לכתוב במחברת הבחינה.
   את התשובות לשאלות 3 ו־4 יש לכתוב **אך ורק** על גבי דף התשובות שבנספח א'.
2. 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 **כטיוטה** (ראשי פרקים, חישובים וכדומה).
כתוב "טיוטה" בראש כל עמוד טיוטה. כתיבת טיוטות כלשהן על דפים שמחוץ למחברת הבחינה עלול לגרום לפסילת הבחינה!

בשאלון זה 47 עמודים ו־3 עמודי נספחים.

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

בהצלחה!

המשך מעבר לדף

השאלות

<u>פרק ראשון</u> (70 נקודות)

ענה על השאלות 1–2 – שאלות חובה.

שאלה 1 – שאלת חובה (40 נקודות)

בשאלה זו שבעה סעיפים (א׳–ז׳). עליך לענות על <u>כל</u> הסעיפים.

מפעל "הבונה" קיבל מספר רב של משימות לביצוע, והוא מעוניין לסדר אותן על־פי עדיפות.

מנהלי המפעל זקוקים למערכת ממוחשבת שתבצע את הפעולות שלהלן:

1. הכנסת משימה למערכת
2. הצגת תיאור מפורט של כל משימה
3. הסרת משימה מן המערכת
4. הדפסת המשימות בסדר יורד על־פי עדיפותן – המשימה בעלת העדיפות הגדולה ביותר תקבל את ערך העדיפות הגדול ביותר, ואילו המשימה בעלת העדיפות הקטנה ביותר תקבל את ערך העדיפות הקטן ביותר.

הטבלה שלהלן מציגה רשימה של משימות שצריך המפעל לבצע. לדוגמה, המשימה שמספרה 80 ועדיפותה **10** נמצאת בעדיפות גדולה יותר מן המשימה שמספרה 20 ועדיפותה **2** .

מספר המשימה	תיאור המשימה	עדיפות
20	A	2
50	B	15
70	C	12
30	D	5
80	E	10
60	F	8
40	G	1

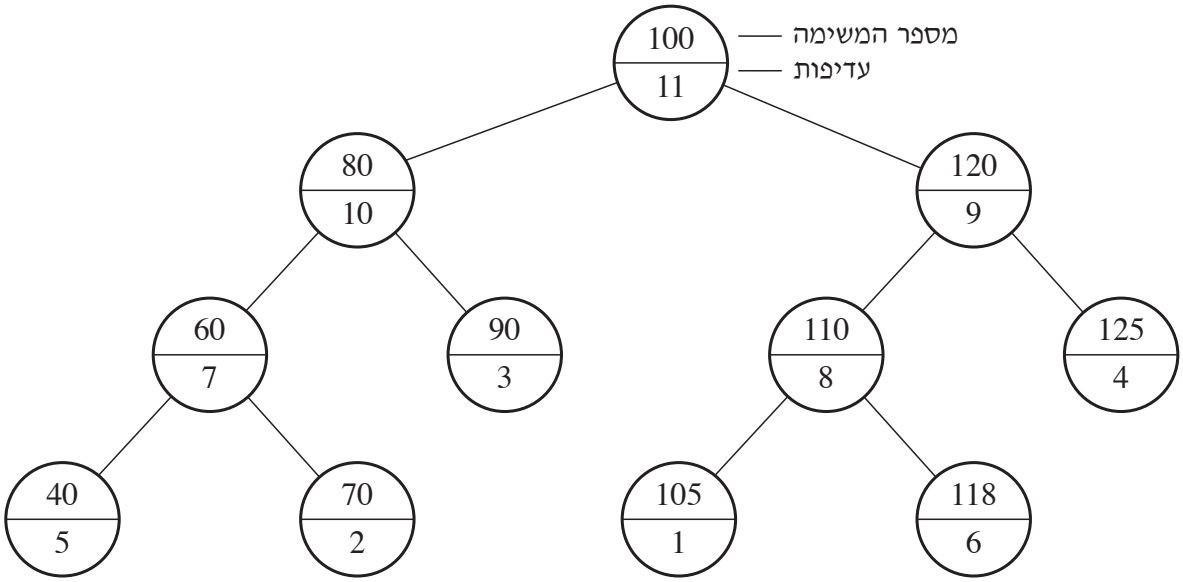
כדי שהמערכת תענה על הדרישות, נשתמש בין היתר במבנה נתונים מיוחד, המכונה: treap .

treap הוא מבנה נתונים המהווה, בעת ובעונה אחת, **עץ חיפוש בינארי** (על־פי מספר המשימה) **וערימת מקסימום** (על־פי עדיפותה).

המשך בעמוד 3

להלן דוגמה למבנה של עץ כזה.

שים לב: עץ זה **אינו** קשור לטבלה המופיעה בעמוד 2.



עץ זה מקיים את תכונותיו של עץ החיפוש הבינארי מבחינת מספר המשימה (מפתח החיפוש), ומקיים את תכונותיה של הערימה מבחינת עדיפות המשימה.

ל־treap יש כמה יתרונות:

1. הוא מהווה עץ חיפוש, מבחינת מספר המשימה.

2. הוא מהווה ערימה, מבחינת עדיפות המשימה.

3. בניית העץ **אינה** תלויה בסדר הכנסתם לעץ של האיברים השונים (בכל סדר שבו נכניס את האיברים, יתקבל תמיד אותו העץ).

הערה: העץ המתקבל הוא לא בהכרח עץ מאוזן.

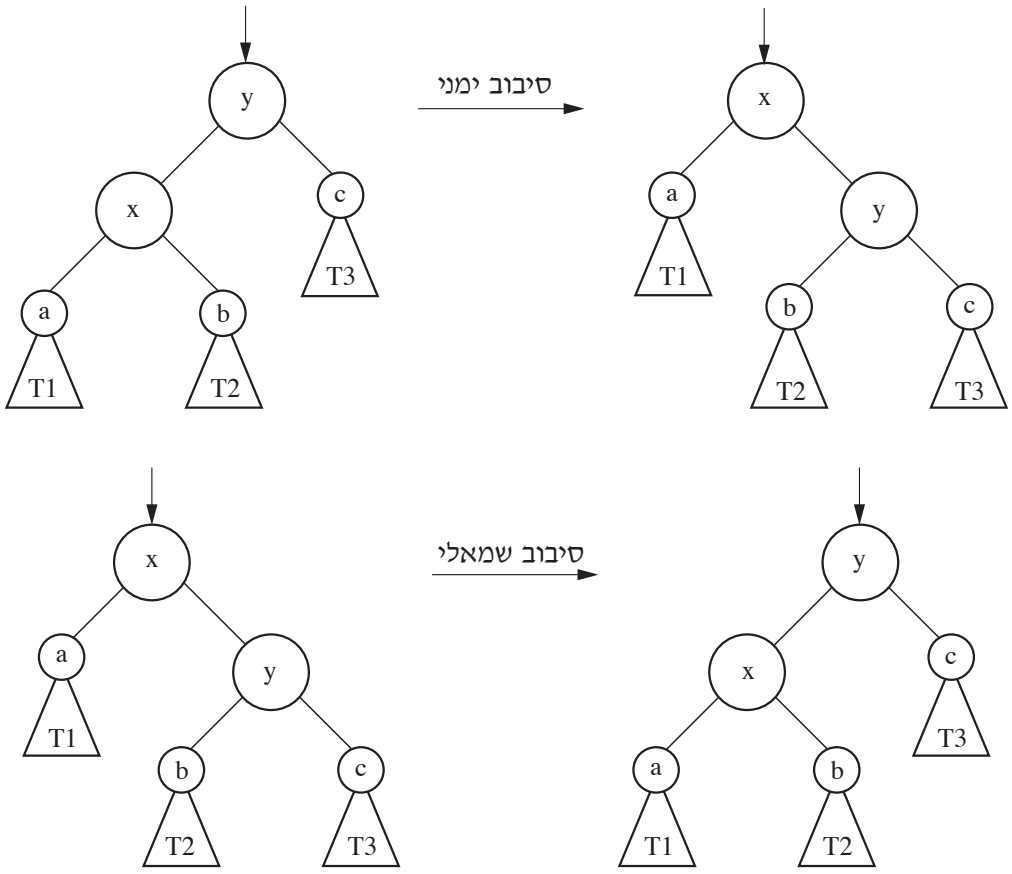
נתון כי לכל צומת בעץ קיימים, בין היתר, שני השדות שלהלן:

key – מספר המשימה, המשמש כמפתח חיפוש בעץ (ומיוצג על־ידי מספר שלם),

value – עדיפות המיוצגת על־ידי מספר שלם. הנח כי העדיפויות **שונות** זו מזו.

המשך בעמוד 4

באיור שלהלן מתוארים סיבוב ימני וסיבוב שמאלי על עץ מטיפוס treap :



להלן אלגוריתם רקורסיבי להכנסת איבר חדש לעץ מטיפוס treap :

נסמן ב־t את המצביע לשורש העץ, ב־key את מספר המשימה המוכנסת לעץ וב־value את עדיפותה.

1. אם t ריק, צור צומת מטיפוס treap שעליו יצביע t , הכנס לתוכו את מספר המשימה (key) ואת עדיפותה (value) , והחזר את t .

2. אם key **קטן** ממספר המשימה של t (ערך שדה ה־key של t), בצע:

2.1 צור צומת חדש עם הנתונים (value , key) , והכנס אותו באופן רקורסיבי לתת־העץ השמאלי.

2.2 אם **הבן השמאלי** של t בעל עדיפות גדולה מזו של t – בצע **סיבוב ימני** סביב הצומת t .

3. אם key **גדול** ממספר המשימה של t , בצע:

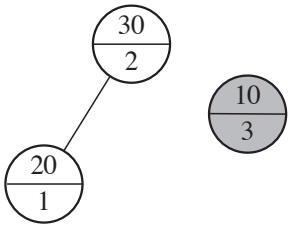
3.1 צור צומת חדש עם הנתונים (value , key) , והכנס אותו באופן רקורסיבי לתת־העץ הימני.

3.2 אם **הבן הימני** של t בעל עדיפות גדולה מזו של t – בצע **סיבוב שמאלי** סביב הצומת t .

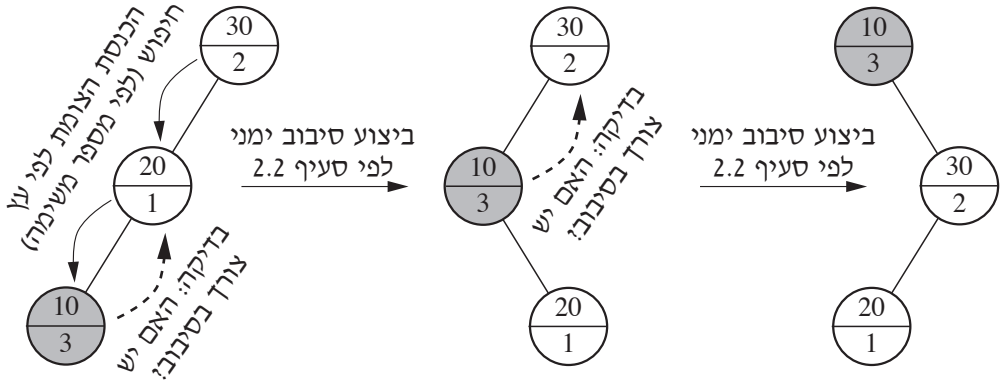
המשך בעמוד 5

לדוגמה:

מעוניינים להכניס את הצומת הצבוע באפור לעץ המופיע משמאלו.



באיור שלהלן מתוארים השינויים החלים בעץ, לאחר הכנסת הצומת, בהתאם לאלגוריתם.



שים לב: לאחר הכנסת צומת, העץ שומר על תכונותיו כעץ חיפוש וכערימה.

(3.5 נק') א. סרטט במחברת הבחינה את העץ המתקבל מהכנסת כל המשימות שבטבלה שבעמוד 2, זו אחר זו, בהתאם לאלגוריתם ההכנסה שבעמוד הקודם.

מעוניינים לתכנן מערכת ממוחשבת שתתמוך, בין היתר, בפעולות שלהלן:

1. **אתחול המערכת – init** – פעולה זו מאתחלת את המערכת, ובין השאר מאפסת את מערך המשימות הדינמי missions , שיפורט בהמשך. הנח כי פעולה זו מתבצעת פעם אחת בלבד.
2. **הוספת משימה למערכת – insertMission** – פעולה זו מוסיפה משימה חדשה למערכת הממוחשבת.
3. **הסרת משימה מן המערכת – deleteMission** – פעולה זו מסירה משימה קיימת מן המערכת.
4. **שיבוץ – schedule** – פעולה זו מוציאה את המשימות מן העץ, מכניסה אותן לתור, ומעדכנת את העץ. המבנים יפורטו בהמשך.
5. **הצגת רשימת המשימות על־פי עדיפותן – printQueue** – פעולה זו מציגה את רשימת המשימות שבתור על־פי סדר העדיפות שלהן.
6. **מציאת משימה – findMission** – פעולה זו מאתרת משימה בעץ ומחזירה את מיקום התא המתאים במערך המשימות.

המשך בעמוד 6

לפניך תיאור של מבנה הנתונים התומך במימוש הפעולות הנדרשות מן המערכת הממוחשבת.

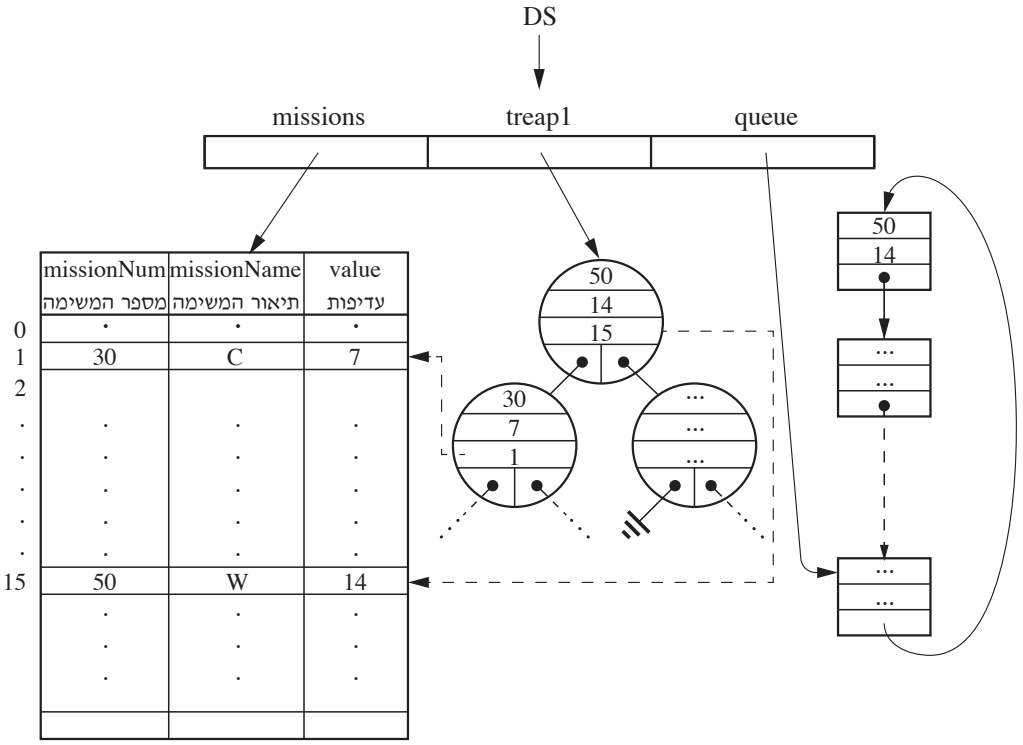
נחזיק מבנה (רשומה), שעליו מצביע DS , ואת המבנה נכנה בשם **"המבנה הראשי"**, המכיל את השדות האלה:

שדה 1: missions – מערך דינמי של משימות, כאשר כל תא במערך מכיל פרטי משימה כלשהי.

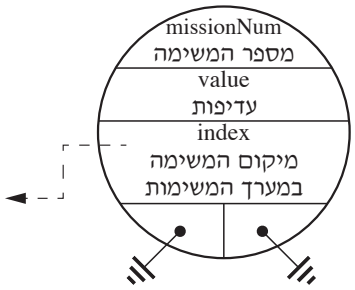
שדה 2: treap1 – מצביע על עץ מטיפוס treap שפורט לעיל.

שדה 3: queue – מצביע לתור שמיוצג על־ידי רשימה מקושרת מעגלית, המשמש בהמשך להצגת המשימות על־פי עדיפותן, בסדר יורד (העדיפות הגדולה תחילה).

באיור שלהלן מוצג התיאור הסכמתי של "המבנה הראשי", המאגד את כל מבני הנתונים שבהם נאחסן את נתוני המערכת הממוחשבת.



מבנה צומת בעץ s מסוג treap :



המשך בעמוד 7

להלן הגדרת המבנה הראשי בשפת C :

```
typedef struct headderType
{
   missionPtr missions;     // מצביע למערך המשימות
   nodePtr treap1;          // מצביע לראשו של עץ מטיפוס treap
   queuePtr queue;          // מצביע לתור
} headder, *headPtr;
```

עתה נפרט את מבנה הנתונים בעבור <u>שדה 1</u> של "המבנה הראשי" – מערך המשימות.

להלן מבנה של תא **במערך המשימות** בשפת C :

```
typedef struct missionType
{
   int missionNum           // מספר המשימה
   char missionName[20];    // תיאור המשימה
   int value;               // עדיפות
} missionRec,*missionPtr;
```

עתה נפרט את מבנה הנתונים בעבור <u>שדה 2</u> של "המבנה הראשי" – עץ מטיפוס treap .

להלן מבנה של צומת **בעץ מטיפוס treap** בשפת C :

```
typedef struct nodeType
{
   int missionNum;          // מספר המשימה
   int value;               // עדיפות
   int index;               // מיקום המשימה במערך המשימות
   struct nodeType *left;   // מצביע לתת־העץ השמאלי
   struct nodeType *right;  // מצביע לתת־העץ הימני
} nodeRec,*nodePtr;
```

המשך בעמוד 8

עתה נפרט את מבנה הנתונים בעבור <u>שדה 3</u> של "המבנה הראשי" – תור המשימות.

להלן מבנה של תא **בתור המשימות** בשפת C :

```
typedef struct queueType
{
   int ID;                    // מספר המשימה
   int value;                 // עדיפות
   struct queueType *next;    // מצביע לצומת הבא
} queueRec,*queuePtr;
```

להלן הגדרות התקפות לכל הסעיפים שיבואו בהמשך:

```
typedef enum {FAILURE,SUCCESS,INVALID_INPUT,RECORD_NOT_FOUND,DUPLICATE_ID} statusType;
typedef enum {FALSE,TRUE} boolean;
```

נתונה ספריית פונקציות המכילה, בין היתר, את הפונקציות שלהלן:

nodePtr **search**(nodePtr root, int key);	פונקצייה זו מקבלת שני פרמטרים: root – מצביע לשורשו של עץ מטיפוס treap , key – מספר המשימה המבוקש. הפונקצייה מחפשת את הצומת שמספר המשימה שלו הוא key בעץ מטיפוס treap , ומחזירה מצביע לצומת המתאים בעץ. אם key לא נמצא בעץ – הפונקצייה תחזיר את הערך NULL .
statusType **findMission**(int key);	פונקצייה זו מקבלת כפרמטר את key , שהוא מספר המשימה. הפונקצייה נעזרת בפונקצייה search שלעיל. אם המשימה נמצאה, הפונקצייה מדפיסה את פרטיה ומחזירה את הערך SUCCESS – אחרת, הפונקצייה מחזירה את הערך RECORD_NOT_FOUND .
queuePtr **insertQueue**(queuePtr q, int key, int value);	פונקצייה זו מקבלת שלושה פרמטרים: q – תור שאליו תוכנס המשימה, key – מספר המשימה, value – עדיפות המשימה. הפונקצייה מכניסה את המשימה לתור ומחזירה מצביע לתור.

הנח שהפונקציות האלו כתובות, וניתן להשתמש בהן בכל הסעיפים הבאים בלי לכתוב אותן מחדש. כמו כן, בעבור כל סעיף תוכל להשתמש בכל פונקצייה שמומשה בסעיפים שלפניו.

המשך בעמוד 9

להלן הגדרות של משתנים גלובליים:

```
headder head;
headPtr DS = &head;   // המבנה הראשי
int lastMission = 0;  // המיקום במערך המשימות שבו תוכנס המשימה הבאה (המיקום מתחיל מ־0)
```

ענה על הסעיפים שלהלן:

(7.5 נק') ב. לצורך סעיף זה, התבונן בשתי הפונקציות שלפניך, המשמשות לביצוע סיבוב ימני וסיבוב שמאלי בעץ.

הפונקציות מתאימות לאיור שמופיע בעמוד 4.

```
nodePtr rightRotate(nodePtr y)
{
    nodePtr x = y->left;
    nodePtr b = x->right;
    x->right = y;
    y->left = b;
    return x;
}

nodePtr leftRotate(nodePtr x)
{
    nodePtr y = x->right;
    nodePtr b = y->left;
    y->left = x;
    x->right = b;
    return y;
}
```

המשך בעמוד 10

לפניך פונקצייה **רקורסיבית** שכותרתה:

```
nodePtr insertTree(nodePtr root, int key, int value, int index)
```

פונקצייה זו מקבלת את הפרמטרים האלה:

root – מצביע לעץ מטיפוס treap ,

key – מספר המשימה שיש להכניס,

value – עדיפות המשימה,

index – מיקום המשימה במערך המשימות.

פונקצייה זו מטפלת בהוספת משימה לעץ מטיפוס treap , ומחזירה מצביע לראש העץ.

הנח כי המשימה שמספרה key אינה קיימת בעץ.

המשך בעמוד 11

בפונקצייה שלהלן 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.

```
nodePtr insertTree(nodePtr root, int key, int value, int index)
{
    if (!root)
     {
      nodePtr t = malloc(sizeof(nodeRec));
      t->missionNum = key;
      t->value = value;
      t->index = index;
      t->left = NULL;
      t->right = NULL;
      return t;
     }
    if (key < root->missionNum)
    {
        root->left = _____(1)_____;
        if (_____(2)_____> root->value)
            root = _____(3)_____;
    }
    else
    {
        root->right = insertTree(root->right, key, value, index);
        if (_____(4)_____)
            root = _____(5)_____;
    }
    return root;
}
```

המשך בעמוד 12

(7.5 נק') ג. לפניך פונקצייה שכותרתה:

```
statusType insertMission(int missionNum, char missionName[], int value)
```

פונקצייה זו מקבלת את הפרמטרים האלה:

missionNum – מספר המשימה,

missionName – תיאור המשימה,

value – עדיפות המשימה.

פונקצייה זו מטפלת בהכנסת משימה למערכת (הן למערך המשימות והן לעץ מטיפוס treap).

הפונקצייה מחזירה ערך מטיפוס statusType כמפורט בטבלה שלהלן:

DUPLICATE_ID	אם המשימה שמספרה missionNum כבר קיימת במערכת הממוחשבת
SUCCESS	אם המשימה שמספרה missionNum נוספה בהצלחה למערכת הממוחשבת

הערה: פונקצייה זו נעזרת בפונקציות הקודמות.

הנח כי אין צורך לבדוק את תקינותן של הקצאות דינמיות בתוכנית.

המשך בעמוד 13

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.

```
statusType insertMission(int missionNum, char missionName[], int value)
{
     statusType status = SUCCESS;
     nodePtr t;
     int index;
     t =  _________(1)_________;
     if (t) return  _________(2)_________;
     if (lastMission == 0)
     {
        DS->missions = malloc(sizeof(missionRec));
     }
     else
     {
        DS->missions = realloc(DS->missions ,  _________(3)_________);
     }
      DS->missions[lastMission].missionNum = missionNum;
      strcpy(DS->missions[lastMission].missionName, missionName);
      DS->missions[lastMission].value = value;
      index =  _________(4)_________;
      lastMission++ ;
      DS->treap1 = _________(5)_________;
      return status;
}
```

המשך בעמוד 14

ד. **(3 נק')** מהי סיבוכיות זמן הריצה של הפונקצייה המוצגת בסעיף ג', אם ידוע ש־n מציין את מספר המשימות במערכת הממוחשבת? כתוב **במחברתך** את התשובה הנכונה.

1. $O(n \log n)$
2. $O(n)$
3. $O(\log n)$
4. $O(n^2)$

ה. **(9 נק')** לפניך פונקצייה **רקורסיבית** שכותרתה:

```
nodePtr deleteNode(nodePtr root, int key)
```

פונקצייה זו מקבלת את הפרמטרים שלהלן:

root – מצביע לעץ מטיפוס treap ,

key – מספר המשימה.

הפונקצייה מסירה מן העץ את הצומת שמספר המשימה שלו הוא key , ומחזירה מצביע לראש העץ.

הפונקצייה מוצאת את הצומת המתאים בעץ על־ידי חיפוש בינארי, ומטפלת בהסרתו באופן הזה:

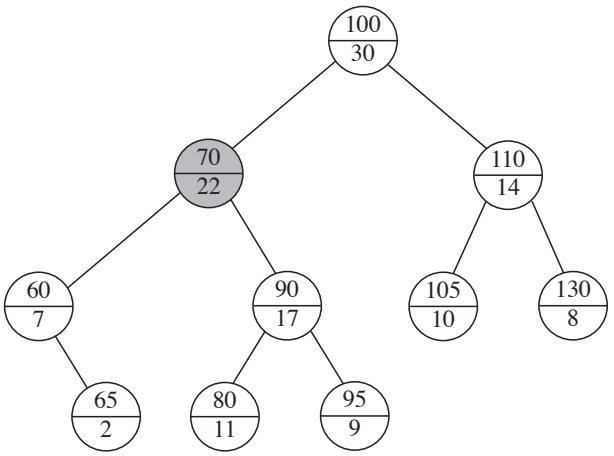
- אם לצומת שמצאנו אין בן שמאלי – המצביע לצומת שמצאנו יצביע לבן הימני שלו.
- אם לצומת שמצאנו אין בן ימני – המצביע לצומת שמצאנו יצביע לבן השמאלי שלו.
- אם לצומת שמצאנו יש שני בנים, הפונקצייה בודקת לאיזה בן יש עדיפות גדולה יותר:

- אם העדיפות של הבן הימני גדולה יותר מן העדיפות של הבן השמאלי – הפונקצייה מבצעת סיבוב שמאלי סביב הצומת שיש להסיר, ומופעלת שוב באופן רקורסיבי על הצומת שיש להסיר.
- אם העדיפות של הבן השמאלי גדולה יותר מן העדיפות של הבן הימני – הפונקצייה מבצעת סיבוב ימני סביב הצומת שיש להסיר, ומופעלת שוב באופן רקורסיבי על הצומת שיש להסיר.

המשך בעמוד 15

לדוגמה:

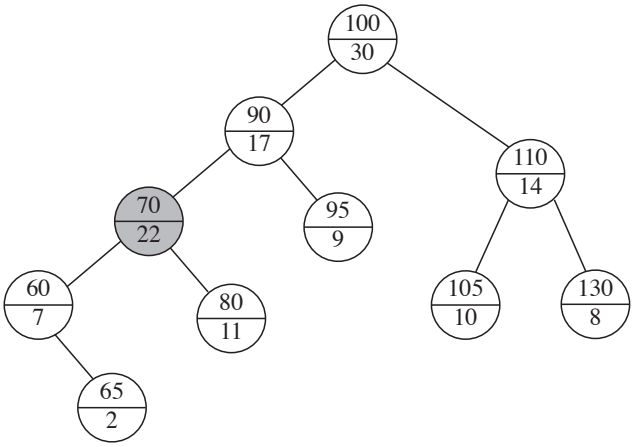
בעבור העץ הזה:



מעוניינים להסיר את הצומת שמכיל את המשימה שמספרה 70.

לצומת יש שני בנים, ועדיפות הבן הימני גדולה מעדיפות הבן השמאלי, ולכן נבצע סיבוב **שמאלי** סביב הצומת הזה.

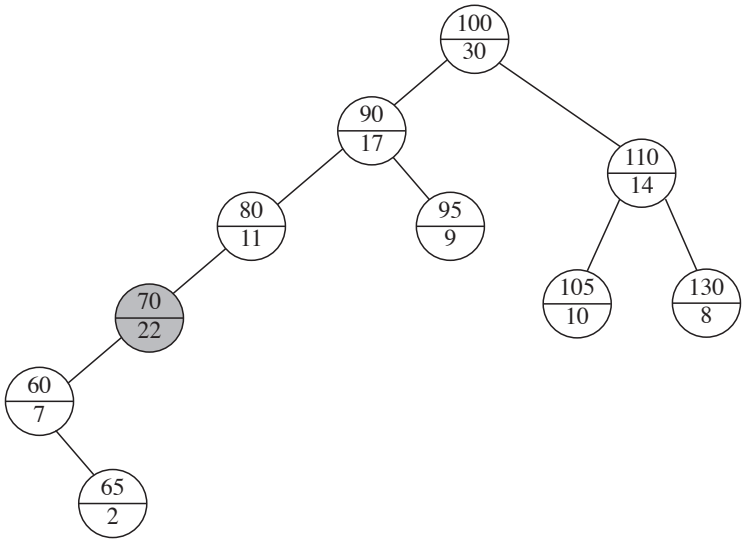
לאחר הסיבוב התקבל העץ הזה:



כיוון שעדיין יש לצומת שני בנים, ועדיפות הבן הימני גדולה מעדיפות הבן השמאלי, נבצע שוב סיבוב **שמאלי** סביבו.

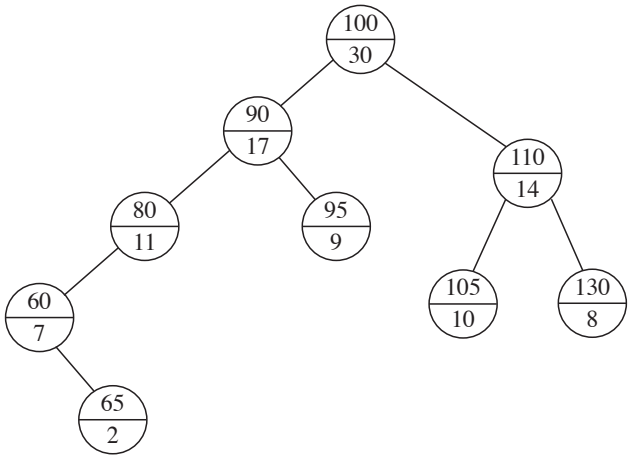
המשך בעמוד 16

לאחר הסיבוב יתקבל העץ הזה:



הפעם, הצומת שמעוניינים להסיר הוא ללא בן ימני, ולכן המצביע לצומת שמעוניינים להסיר יצביע לבנו השמאלי.

לבסוף, קיבלנו את העץ שלהלן, ללא הצומת.



המשך בעמוד 17

בפונקצייה חסרים **שישה** ביטויים, המסומנים במספרים בין סוגריים עגולים. כתוב במחברת הבחינה את מספרי הביטויים החסרים (1)—(6) , בסדר עולה, וכתוב ליד כל מספר את הביטוי החסר שהוא מייצג.

```
nodePtr deleteNode(nodePtr root, int key)
{
    nodePtr temp;
    if (root == NULL)
        return root;
    if (key < root->missionNum)
        root->left =  _________(1)_________;
    else if (key > root->missionNum)
        root->right =  _________(2)_________;
    else if (root->left == NULL)
    {
        temp =  _________(3)_________;
        free(root);
        root = temp;
    }
    else if (root->right == NULL)
    {
        temp =  _________(4)_________;
        free(root);
        root = temp;
    }
    else if (root->left->value < root->right->value)
    {
        root = leftRotate(root);
        root->left =  _________(5)_________;
    }
    else
    {
        root = rightRotate(root);
        root->right =  _________(6)_________;
    }
    return root;
}
```

המשך בעמוד 18

(7.5 נק') ו. לפניך פונקצייה שכותרתה:

```
statusType schedule(nodePtr t)
```

פונקצייה זו מקבלת את הפרמטר t , שהוא מצביע לעץ מטיפוס treap .

הפונקצייה יוצרת תור עם עדיפויות.

הפונקצייה מחזירה ערך מטיפוס statusType , כמפורט בטבלה שלהלן:

FAILURE	אם העץ שעליו מצביע t הוא ריק
SUCCESS	אם המשימה בוצעה בהצלחה

הפונקצייה מתייחסת לעץ כאל ערימה. היא מוציאה את איברי הערימה ומעבירה אותם לתור. לאחר מכן היא משחזרת מחדש את העץ מתוך מערך המשימות.

פונקצייה זו נעזרת בפונקציות הקודמות.

המשך בעמוד 19

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.

```
statusType schedule(nodePtr t)
{
   int key;
   int value;
   int i, index;
   statusType status = SUCCESS;
   if(t == NULL) return FAILURE;
   while(t)
   {
      key = t->missionNum;
      value = t->value ;
      DS->queue = _________(1)_________;
      t = _________(2)_________;
   }
   DS->treap1 = NULL;
   for(i=0; i < _________(3)_________; i++)
   {
      key = DS->missions[i].missionNum;
      value = DS->missions[i].value;
      index = _________(4)_________;
      DS->treap1 = _________(5)_________;
   }
   return status;
}
```

המשך בעמוד 20

ז. **(2 נק')** מהי סיבוכיות זמן הריצה של הפונקצייה המוצגת בסעיף ו', אם ידוע ש־n מציין את מספר המשימות במערכת הממוחשבת?

כתוב **במחברתך** את התשובה הנכונה.

1. $O(n^2)$
2. $O(\log n)$
3. $O(n \log n)$
4. $O(n)$

המשך בעמוד 21

שאלה 2 – שאלת חובה (30 נקודות)

בשאלה זו שמונה סעיפים (א'–ח'). עליך לענות על <u>כל</u> הסעיפים.

אנו מעוניינים ליצור מילון אלקטרוני לביאור מונחים נפוצים במדעי המחשב. קיימות כמה אפשרויות ליצירת מילון אלקטרוני כזה.

אפשרות א':

קולטים לתוך מערך את המונחים ואת פירושם, וממיינים את המערך לפי סדר אלפבתי באמצעות אלגוריתם למיון מהיר.

נניח כי המילון מכיל n מונחים.

(2 נק') **א.** מהי סיבוכיות זמן הריצה ליצירת המילון?

כתוב **במחברתך** את התשובה הנכונה.

1. $O(n^2)$
2. $O(n)$
3. $O(n \log n)$
4. $O(n^2 \log n)$

(2 נק') **ב.** מהי סיבוכיות זמן הריצה להוספת מונח חדש למילון?

כתוב **במחברתך** את התשובה הנכונה.

1. $O(n)$
2. $O(\log n)$
3. $O(n \log n)$
4. $O(n^2)$

(2 נק') **ג.** מהי סיבוכיות זמן הריצה להסרת מונח מן המילון?

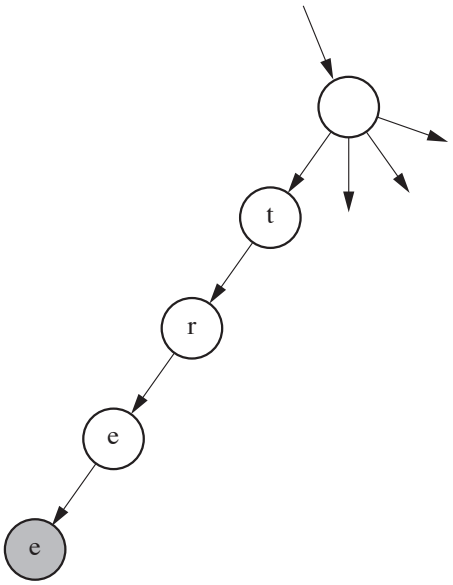
כתוב **במחברתך** את התשובה הנכונה.

1. $O(\log n)$
2. $O(n^2)$
3. $O(n \log n)$
4. $O(n)$

המשך בעמוד 22

אפשרות ב':

נשתמש בגרף מיוחד הדומה לעץ, שבו כל מונח מיוצג על־ידי רשימה מקושרת של האותיות המרכיבות אותו.
למשל, המונח tree יאוחסן באופן הזה:



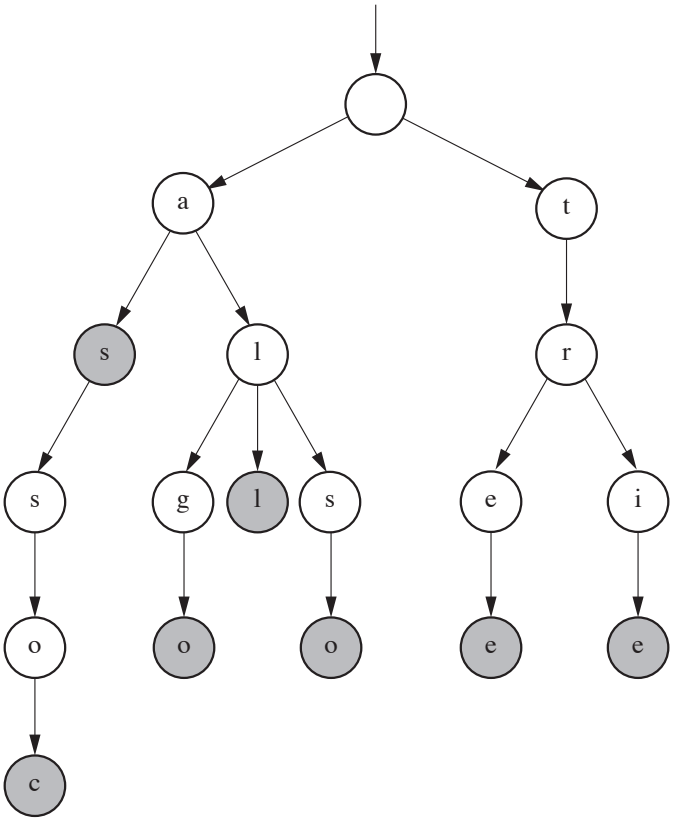
הצומת המסומן באפור הוא הצומת האחרון במונח tree . הוא מכיל את האות האחרונה של המונח, וגם את פירושו.

בטבלה שלהלן מוצגת רשימה של מונחים ופירושם. לדוגמה:

המונח	הפירוש
as	like
assoc	association
algo	algorithm
all	total
also	decision
tree	data structure
trie	graph

המשך בעמוד 23

להלן דוגמה לגרף המייצג את המילון של המונחים: as, assoc, algo, all, also, tree, trie .



שים לב:

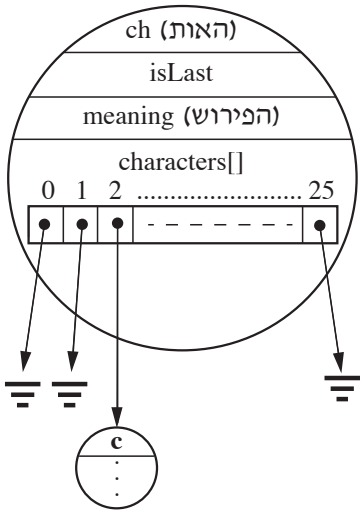
1. שורש הגרף **אינו** מכיל שום אות.
2. **הצומת האפור, המכיל את האות s** , משמש צומת רגיל בעבור המונח assoc אך גם מהווה צומת אחרון בעבור המונח as ומכיל גם את פירושו.

המשך בעמוד 24

להלן תיאור המבנה של צומת בגרף:

1. המשתנה ch , המכיל אות אחת מבין 26 האותיות האלפבית האנגלי;
2. המשתנה הבוליאני isLast , המציין אם האות בצומת הזה היא האות האחרונה של המונח (TRUE) או לא (FALSE) . למשל, בדוגמה המובאת בעמוד הקודם הצומת המודגש באפור, אשר מכיל את האות s , מהווה גם את סוף המונח as , ולכן – למרות היותו חלק ממונח אחר – ערך ה־isLast שלו יהיה TRUE .
3. המשתנה meaning , המכיל את פירוש המונח; בצומת המהווה אמצע מונח הפירוש יקבל ערך ריק.
4. שדה קישור – characters[] – מערך של 26 מצביעים, כמספר האותיות באלפבית האנגלי, הגורמים לכך שלכל צומת בגרף יכולים להיות 26 בנים (לכל היותר).

להלן איור המתאר את המבנה של צומת בגרף:

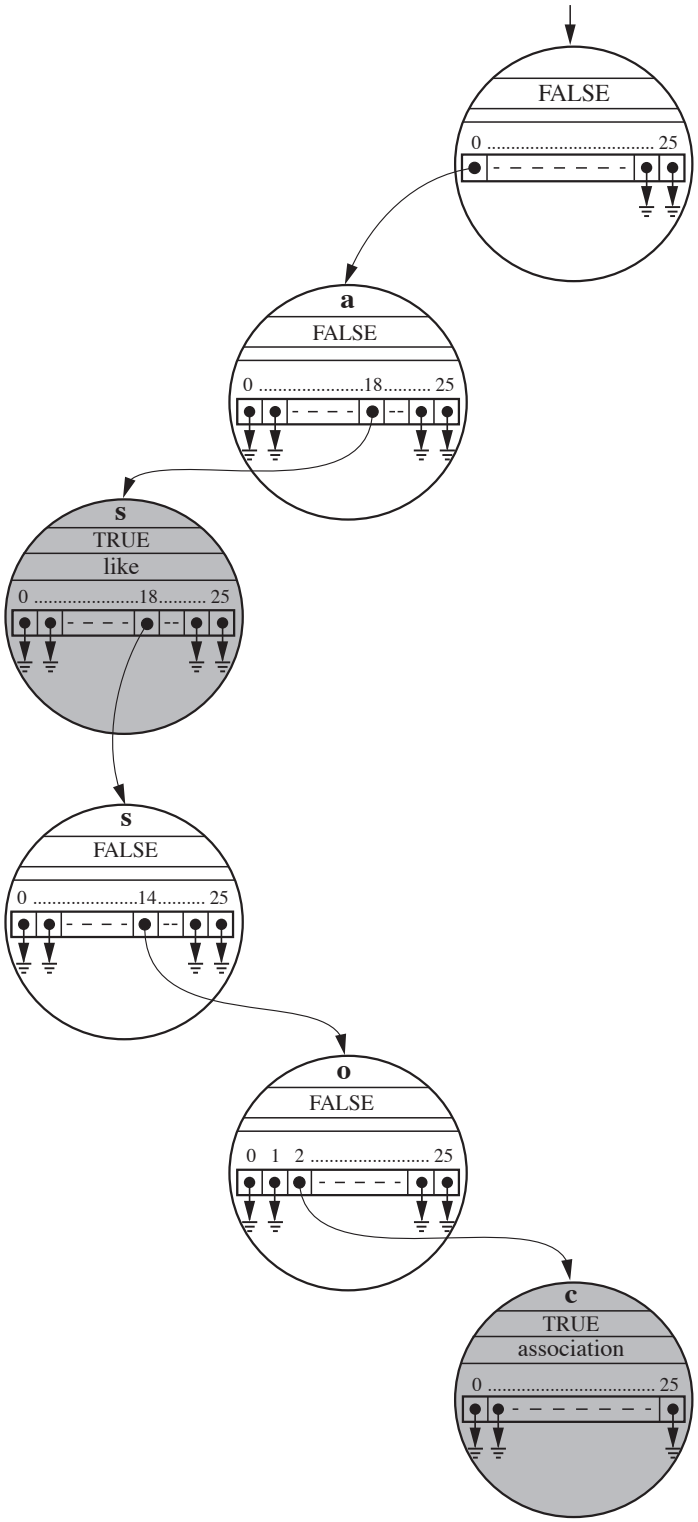


מיקום המצביע במערך המצביעים characters[] שווה למיקום האות באלפבית יחסית למיקום 'a' , שהוא למעשה ההפרש בין ערכי האסקי של התווים.

למשל, מיקום המצביע שיתאים לאות c יימצא במקום 'c'–'a'=2 במערך המצביעים של הצומת הקודם. הקישור לצומת הבא בעץ יהיה על־פי המצביע במערך characters[] של הצומת הקודם.

המשך בעמוד 25

המונחים assoc ו־as יאוחסנו במילון באופן הזה:



המשך בעמוד 26

להלן הגדרות של טיפוסי הנתונים שבהם התוכנית עושה שימוש:

```
#define ALPHABET_SIZE 26

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

להלן הגדרת המבנה של צומת בעץ:

```
typedef struct trieType
{
   char ch; // האות
   boolean isLast;       // משתנה המציין אם האות מהווה סוף מונח
   char meaning[40]; // פירוש המונח
   struct trieType *characters[ALPHABET_SIZE]; // מערך מצביעי האותיות
} trieRec, *triePtr;


triePtr head; // משתנה גלובלי המצביע לראש הגרף
```

להלן הפונקצייה `init()` הבונה את שורש הגרף, שכאמור אינו מכיל שום אות:

```
void init()
{
   int i;
   head = (triePtr)malloc(sizeof(trieRec));
   head->isLast = FALSE;
   for(i= 0;i < ALPHABET_SIZE;i++)
         head->characters[i]= NULL;
}
```

המשך בעמוד 27

ד. לפניך פונקצייה שכותרתה: **(6 נק')**

```
void insert(triePtr head, char *str, char *meaning)
```

פונקצייה זו מטפלת בהוספת מונח למילון (מילה). הנח כי המונח אינו קיים במילון.

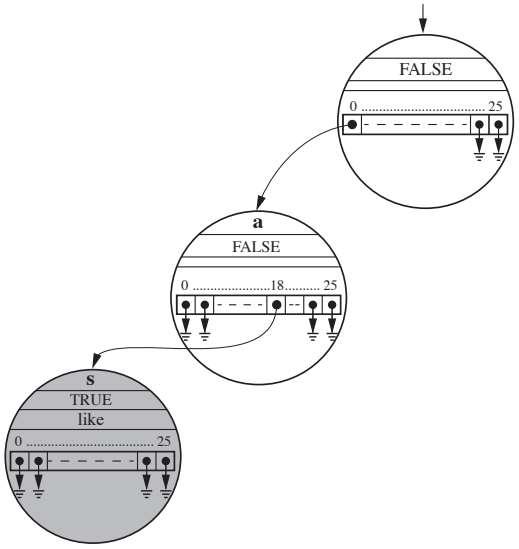
פונקצייה זו מקבלת את הפרמטרים האלה:

head – מצביע לראש הגרף,

str – מונח חדש שיש להוסיף למילון,

meaning – פירוש המונח.

הפונקצייה עוברת על כל אות במונח, לפי הסדר, ומכניסה אותה לגרף, כמתואר באיור שלהלן:



על־פי האיור, הגרף מכיל רק את המונח as , ומעוניינים להוסיף לגרף גם את המונח all .

הפונקצייה סורקת את המונח all , אות אחר אות.

מכיוון שהאות a כבר קיימת בגרף כחלק מן המונח as , כלומר בצומת השורש, ערכו של המצביע במקום ה־0 במערך []characters אינו NULL . לפיכך, אין צורך להוסיפה, ויש להתקדם לאות הבאה אחריה במונח, שהיא האות *l* .

כעת, בעבור הצומת שמכיל את האות a , ערכו של המצביע במקום ה־'l'–'a'=11 במערך []characters הוא NULL . משמע: אין במילון מונח שמתחיל בצמד האותיות 'al' . לכן יוצרים צומת חדש המכיל את האות *l* , ומתקדמים לאות הבאה אחריה במונח, שהיא האות *l* .
בצומת האחרון במונח, ערכו של המשתנה isLast יהיה TRUE , והצומת יכיל את פירוש המונח.

אם צריך להוסיף למילון מונח שהוא תת־מילה של מילה שכבר קיימת בגרף, יש לשנות את ערכו של המשתנה isLast של הצומת לערך TRUE , גם אם הצומת מהווה את אמצעו של מונח אחר, ארוך יותר.

המשך בעמוד 28

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(triePtr head, char *str, char *meaning)
{
   triePtr t;
   int i;
   triePtr curr = head;
   while (*str)
   {
       if (curr->characters[_________(1)_________] == NULL)
       {
           t = (triePtr)malloc(sizeof(trieRec));
           t->ch = str[0];
           _________(2)_________ = FALSE;
           for(i=0; i < ALPHABET_SIZE; i++)
               t->characters[i]= NULL;
           curr->characters[*str – 'a'] = _________(3)_________;
       }
       curr = curr->characters[*str – 'a'];
       _________(4)_________;
   }
   strcpy(curr->meaning, meaning);
   _________(5)_________;
}
```

המשך בעמוד 29

(2 נק') **ה.** מהי סיבוכיות זמן הריצה של הפונקצייה insert , אם ידוע שהגרף מכיל n מונחים ואורכו של המונח הארוך ביותר הוא H ?

כתוב **במחברתך** את התשובה הנכונה.

1. $O(n * H)$
2. $O(H \log n)$
3. $O(H \log H)$
4. $O(H)$

(8 נק') **ו.** לפניך פונקצייה שכותרתה:

```
char *search(triePtr head, char* str)
```

פונקצייה זו מקבלת את הפרמטרים האלה:

head – מצביע לראש הגרף,

str – המונח שאותו מחפשים.

הפונקצייה מאתרת בגרף את המונח ומחזירה את פירושו. אם המונח המבוקש לא נמצא במילון, הפונקצייה תחזיר את המחרוזת: WORD NOT FOUND .

הפונקצייה סורקת את המונח, אות אחר אות, ומחפשת בגרף מסלול התואם את אותיות המונח.

מונח לא יימצא בגרף (במילון) במקרים שלהלן:

- לא נמצא בגרף מסלול התואם את המילה המבוקשת (המונח לא קיים בגרף).
- קיים בגרף מסלול המסתיים לפני סוף המילה המבוקשת (קיימת בגרף תת־מילה של המילה המבוקשת).
- קיים בגרף מסלול המתאים למונח כולו, אבל האות האחרונה במסלול אינה מהווה סוף מילה בגרף.

המשך בעמוד 30

בפונקצייה חסרים **שלושה** ביטויים, המסומנים במספרים בין סוגריים עגולים. כתוב במחברת הבחינה את מספרי הביטויים החסרים (1)—(3) , בסדר עולה, וכתוב ליד כל מספר את הביטוי החסר שהוא מייצג.

```
char *search(triePtr head, char* str)
{
    triePtr curr;
    _________(1)_________;
    while (*str)
    {
        curr = _________(2)_________;
        if (curr == NULL)
            return "WORD NOT FOUND \n";
        str++;
    }
    if (_________(3)_________)
        return (curr->meanning);
    else
        return "WORD NOT FOUND \n";
}
```

ז. (2 נק') מהי סיבוכיות זמן הריצה של הפונקצייה search , אם ידוע שהגרף מכיל n מונחים ואורכו של המונח הארוך ביותר הוא H ?

כתוב **במחברתך** את התשובה הנכונה.

1. $O(n)$
2. $O(H \log n)$
3. $O(H)$
4. $O(\log H)$

המשך בעמוד 31

ח. (6 נק') לצורך הפונקצייה delete , שמטרתה להסיר מונח מן הגרף בצורה בטוחה, נגדיר את הפונקצייה שכותרתה:

```
boolean haveChildren(triePtr curr)
```

פונקצייה זו מקבלת כפרמטר את curr , שהוא מצביע לצומת מסוים בגרף, ומחזירה TRUE אם לצומת יש לפחות בן אחד – אחרת, הפונקצייה מחזירה FALSE .

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.

```
boolean haveChildren(triePtr curr)
{
   int i;
   for (i=0; i < __________(1)__________; i++)
       if (__________(2)__________)
           return TRUE;
   return FALSE;
}
```

המשך בעמוד 32

פרק שני (30 נקודות)

ענה על אחת מבין השאלות 3–4 (לכל שאלה – 30 נקודות).

שאלה 3 (30 נקודות)

בשאלה זו 12 סעיפים. עליך לענות על כל הסעיפים (לכל סעיף – 2.5 נקודות).
בכל סעיף נתונות ארבע תשובות, שרק אחת מהן נכונה. בחר את התשובה הנכונה, והקף בעיגול את הספרה המייצגת אותה בדף התשובות שבנספח א'.

בסעיפים א'–ה' התייחס לבעיה שלהלן:

נתונה מחסנית ונתונות פונקציות לטיפול במחסנית: top , pop , push , isEmpty .

להלן הגדרת מבנה של איבר במחסנית:

```
struct stack
{
  int data;
  struct stack * next;
};
```

מטרת התוכנית היא למיין את איברי המחסנית.

לצורך כך, נגדיר פונקצייה **רקורסיבית** שכותרתה:

```
void sortedInsert(struct stack **s, int x)
```

פונקצייה זו מקבלת מצביע לאיבר הנמצא בראש מחסנית ממוינת כלשהי, ואיבר כלשהו נוסף. הפונקצייה מכניסה את האיבר למחסנית, כך שהמחסנית תישאר ממוינת (האיבר הגדול ביותר נמצא בראש המחסנית).

המשך בעמוד 33

בפונקצייה חסרים **שלושה** ביטויים, המסומנים במספרים בין סוגריים עגולים. בכל אחד מן הסעיפים שלהלן, בחר את הביטוי החסר מבין ארבע האפשרויות הנתונות, והקף בעיגול את הספרה המייצגת אותו בדף התשובות שבנספח א'.

```
void sortedInsert(struct stack **s, int x)
{
    int temp;
    if (_________(1)_________)
    {
       push(s, x);
       return;
    }
    temp = pop(s);
    _________(2)_________;
    _________(3)_________;
}
```

א. הביטוי החסר (1) הוא:

1. `isEmpty(*s)`
2. `x > top(*s)`
3. `isEmpty(*s) || x > top(*s)`
4. `!isEmpty(*s)`

ב. הביטוי החסר (2) הוא:

1. `push(s, x)`
2. `sortedInsert(s, x)`
3. `if(isEmpty(*s))`
4. `push(s, temp)`

ג. הביטוי החסר (3) הוא:

1. `sortedInsert(s, x)`
2. `push(s, x)`
3. `return`
4. `push(s, temp)`

המשך בעמוד 34

להלן פונקצייה **רקורסיבית** נוספת שכותרתה:

```
void sortStack(struct stack **s)
```

פונקצייה זו מקבלת מחסנית וממיינת אותה. הפונקצייה נעזרת בפונקציות שהוזכרו לעיל.

בפונקצייה חסרים **שני** ביטויים, המסומנים במספרים בין סוגריים עגולים. בכל אחד מן הסעיפים שלהלן, בחר את הביטוי החסר מבין ארבע האפשרויות הנתונות, והקף בעיגול את הספרה המייצגת אותו בדף התשובות שבנספח א'.

```
void sortStack(struct stack **s)
{
    int x;
    if (!isEmpty(*s))
    {
        x = pop(s);
        _________(1)_________;
        _________(2)_________;
    }
}
```

ד. הביטוי החסר (1) הוא:

1. `sortedInsert(s, x)`
2. `sortStack(s)`
3. `push(s, x)`
4. `sortedInsert(*s, x)`

ה. הביטוי החסר (2) הוא:

1. `sortedInsert(s, x)`
2. `sortStack(s)`
3. `push(s, x)`
4. `sortedInsert(*s, x)`

המשך בעמוד 35

סעיפים ו'–ז' מתייחסים לבעיה שלהלן:

נתון מערך A המכיל n מספרים שלמים, ונתון מספר שלם k , כאשר k<n .

אנו מעוניינים למצוא את k האיברים הגדולים במערך A .

להלן אלגוריתם לביצוע המשימה:

1. צור מערך עזר Temp בגודל k והעבר לתוכו את k האיברים **הראשונים** של A .

2. עבור i , כאשר: i = k ..... n–1 , בצע:

2.1 מצא את מיקומו של האיבר הקטן ביותר במערך Temp והכנס אותו למשתנה min .

2.2 אם Temp[min] < A[i] החלף את Temp[min] ב־A[i] .

3. הדפס את המערך Temp .

ו. מהי סיבוכיות זמן הריצה של האלגוריתם?

1. $O(n^2)$
2. $O(k^2)$
3. $O(n*k)$
4. $O((n-k)*k)$

ז. אם מעוניינים לקבל את התוצאה שקיבלנו מביצוע האלגוריתם וגם למיין את המערך Temp , כך שהאיבר הקטן ביותר יהיה במקום ה־0 , מה תהיה אז סיבוכיות זמן הריצה של האלגוריתם?

1. $O(n \log n)$
2. $O((n-k)*k \log k)$
3. $O((n-k)*k + k \log k)$
4. $O(n \log k)$

המשך בעמוד 36

סעיפים ח'–י"ב מתייחסים לבעיה שלהלן:

להלן הגדרת מבנה של צומת בעץ בינארי נתון:

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


typedef struct nodeType
{
    int data;
    struct nodeType *left;
    struct nodeType *right;
}nodeRec ,*nodePtr;
```

להלן פונקצייה **רקורסיבית** שכותרתה:

```
boolean isBST(nodePtr node)
```

פונקצייה זו מקבלת עץ בינארי, שאינו בהכרח עץ מאוזן, ובודקת אם העץ הוא עץ חיפוש בינארי (BST) .

הפונקצייה מחזירה TRUE אם העץ הוא BST – אחרת, היא מחזירה FALSE .

הפונקצייה נעזרת בשתי פונקציות עזר: int maxValue(nodePtr p) שזמן ריצתה הוא $O(n)$.

int minValue(nodePtr p) שזמן ריצתה הוא $O(n)$.

פונקציות אלה מקבלות מצביע לשורש העץ, ומחזירות את המספר הגדול או את המספר הקטן בעץ, בהתאמה.

המשך בעמוד 37

בפונקצייה חסרים **ארבעה** ביטויים, המסומנים במספרים בין סוגריים עגולים. בכל אחד מן הסעיפים שלהלן, בחר את הביטוי החסר מבין ארבע האפשרויות הנתונות, והקף בעיגול את הספרה המייצגת אותו בדף התשובות שבנספח א'.

```
boolean isBST(nodePtr node)
{
  if (node == NULL)
      return(TRUE);
  if (node->left != NULL && _____(1)_____)
      return(FALSE);
  if (node->right != NULL && _____(2)_____)
      return(FALSE);
  if (_____(3)_____ || _____(4)_____)
      return(FALSE);
  return(TRUE);
}
```

ח. הביטוי החסר (1) הוא:

1. `node->left->data > node->data`
2. `maxValue(node) > node->data`
3. `maxValue(node->left) > node->data`
4. `node->data > node->left->data`

ט. הביטוי החסר (2) הוא:

1. `node->right->data < node->data`
2. `minValue(node) < node->data`
3. `minValue(node->right) < node->data`
4. `node->data < node->right`

המשך בעמוד 38

י. הביטוי החסר (3) הוא:

1. `!isBST(node)`
2. `!isBST(node->left)`
3. `isBST(node)`
4. `isBST(node->left)`

י"א. הביטוי החסר (4) הוא:

1. `!isBST(node)`
2. `!isBST(node->right)`
3. `isBST(node)`
4. `isBST(node->right)`

י"ב. מהי סיבוכיות זמן הריצה של הפונקצייה isBST ?

1. $O(n^2)$
2. $O(n \log n)$
3. $O(n)$
4. $O(n^2 \log n)$

המשך בעמוד 39

שאלה 4 (30 נקודות)

בשאלה זו 12 סעיפים. עליך לענות על כל הסעיפים (לכל סעיף – 2.5 נקודות).
בכל סעיף נתונות ארבע תשובות, שרק אחת מהן נכונה. בחר את התשובה הנכונה, והקף בעיגול את הספרה המייצגת אותה בדף התשובות שבנספח א'.

בסעיפים א'-ה' התייחס לבעיה שלהלן:

מעוניינים ליצור תוכנית המקבלת כפרמטר מספר n שלם וחיובי, ומדפיסה את כל המספרים מ־1 עד n (כולל 1 ו־n) בתצוגה בינארית, תוך שימוש בתור המיושם על־ידי רשימה מקושרת מעגלית.

הערה: ברשימה מקושרת מעגלית האיבר האחרון מצביע לראש הרשימה.

להלן הגדרת מבנה התור:

```
typedef struct queueType
{
   char data[20];
   struct queueType *next;
}queueRec ,*queue;
```

התוכנית נעזרת בפונקצייה שכותרתה:

```
void insertQ(queue *que, char x[])
```

פונקצייה זו מקבלת תור וסדרה בינארית כמחרוזת, ומכניסה את המחרוזת כולה לתוך התור.

המשך בעמוד 40

בפונקצייה חסרים **שני** ביטויים, המסומנים במספרים בין סוגריים עגולים. בכל אחד מן הסעיפים שלהלן, בחר את הביטוי החסר מבין ארבע האפשרויות הנתונות, והקף בעיגול את הספרה המייצגת אותו בדף התשובות שבנספח א'.

```
void insertQ(queue *que, char x[])
{
   queue p = malloc(sizeof(queueRec));
   strcpy(p->data, x);
   if (*que == NULL)
   {
      *que = p;
      (*que)->next = *que;
   }
   else
   {
       _________(1)_________;
       _________(2)_________;
   }
   *que = p;
}
```

א. הביטוי החסר (1) הוא:

1. `p = (*que)->next`
2. `p->next = (*que)`
3. `p->next = (*que)->next`
4. `p->next = (que)->next`

ב. הביטוי החסר (2) הוא:

1. `(que)->next = p`
2. `(*que)->next = p`
3. `(*que)->next = *p`
4. `(*que) = p->next`

המשך בעמוד 41

להלן פונקצייה שכותרתה:

```
void generatePrintBinary(int n)
```

פונקצייה זו מקבלת כפרמטר מספר שלם n , ומדפיסה את כל המספרים מ־1 עד n בתצוגה בינארית.

הפונקצייה נעזרת גם בפונקצייה char * deleteQ(queue * que) , שהיא פונקצייה קיימת, אשר מסירה איבר מתור המיושם על־ידי רשימה מקושרת מעגלית, ומחזירה את ערך האיבר שבראש התור (q->next->data) .

הערה: בכל איטרציה, המספר הראשון שנמצא בראש התור משמש ליצירת שני המספרים הבינאריים הבאים, הנוצרים במשתנים s1 ו־s2 על־ידי שרשור 0 ו־1 בהתאמה.

בפונקצייה חסרים **שלושה** ביטויים, המסומנים במספרים בין סוגריים עגולים. בכל אחד מן הסעיפים שלהלן, בחר את הביטוי החסר מבין ארבע האפשרויות הנתונות, והקף בעיגול את הספרה המייצגת אותו בדף התשובות שבנספח א'.

```
void generatePrintBinary(int n)
{
    queue q = NULL;
    char s1[20];
    char s2[20];
    insertQ(&q, "1");
    while (n--)
    {
       printf("\n%s", q->next->data);
       strcpy(_________(1)_________);
       strcpy(s2, s1);
       strcat(s1, "0");
       _________(2)_________;
       strcat(s2, "1");
       _________(3)_________;
    }
}
```

המשך בעמוד 42

ג. הביטוי החסר (1) הוא:

1. `s1, q->data`
2. `s2, s1`
3. `s1, "0"`
4. `s1, deleteQ(&q)`

ד. הביטוי החסר (2) הוא:

1. `strcpy(s1, deleteQ(&q))`
2. `strcpy(s2, deleteQ(&q))`
3. `insertQ(&q, s1)`
4. `insertQ(&q, s2)`

ה. הביטוי החסר (3) הוא:

1. `strcpy(s1, deleteQ(&q))`
2. `strcpy(s2, deleteQ(&q))`
3. `insertQ(&q, s1)`
4. `insertQ(&q, s2)`

סעיפים ו'–ח' אינם תלויים זה בזה.

ו. $T(n)=4T(n/2)+7n^2$ היא פונקציית זמן הריצה של אלגוריתם מסוים, הפועל על קלט שגודלו n .

מהי סיבוכיות זמן הריצה של האלגוריתם?

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

המשך בעמוד 43

ז. $T(n)=3T\left(\frac{n}{3}\right)+\sqrt{n}$ היא פונקציית זמן הריצה של אלגוריתם מסוים, הפועל על קלט שגודלו n .
מהי סיבוכיות זמן הריצה של האלגוריתם?

1. $\Theta(n)$
2. $\Theta(2^n)$
3. $\Theta(n^2)$
4. $\Theta(n^{log3})$

ח. נתון קטע קוד:

```
void func5(int n)
{
   int i, j, temp, m;
   temp = n;
   m = n ;
   while (temp != 0)
    {
       n = n + m;
       temp = temp/2;
    }
   for (i = 0; i < n; i++)
      for (j = 0; j < 2019; j++)
          printf("hi!");
}
```

בהתייחס לערך ההתחלתי של n , מהי סיבוכיות זמן הריצה של קטע הקוד הנתון כפונקצייה של n ?

1. $\Theta(n\,log\,n)$
2. $\Theta(log\,n)$
3. $\Theta(n^2)$
4. $\Theta(n)$

המשך בעמוד 44

בסעיפים ט'–י"ב התייחס לבעיה שלהלן:

נתונה G[V][V] שהיא מטריצת סמיכויות של גרף מכוון כלשהו.

מעוניינים לכתוב תוכנית המקבלת כפרמטר את מטריצת הסמיכויות של הגרף, ובונה בעזרתה את הגרף, כאשר הגרף מיוצג על־ידי רשימות מקושרות (רשימת סמיכויות).

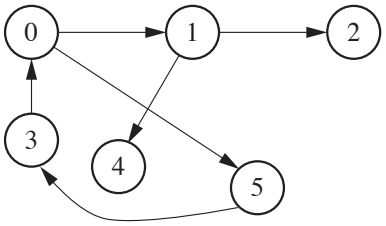
להלן הגדרה של מבנה צומת ברשימה מקושרת, ומערך מצביעים:

```
typedef struct listType
{
 int num;
 struct listType *next;
}listRec, *list;

list vertices[V];
```

לדוגמה:

בעבור הגרף הזה:

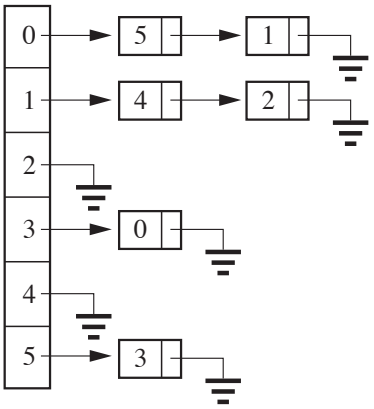


הפונקצייה מקבלת את מטריצת הסמיכויות G שלהלן:

	0	**1**	**2**	**3**	**4**	**5**
0	0	1	0	0	0	1
1	0	0	1	0	1	0
2	0	0	0	0	0	0
3	1	0	0	0	0	0
4	0	0	0	0	0	0
5	0	0	0	1	0	0

המשך בעמוד 45

ובונה בעבורו את המערך vertices שלהלן:



להלן פונקצייה שכותרתה:

```
void build(int G[][V])
```

פונקצייה זו מקבלת כפרמטר את G , שהיא מטריצת הסמיכויות של הגרף, ובונה בעזרתה את הגרף, כאשר הגרף מיוצג על־ידי רשימות מקושרות.

המשך בעמוד 46

בפונקצייה חסרים **ארבעה** ביטויים, המסומנים במספרים בין סוגריים עגולים. בכל אחד מן הסעיפים שלהלן, בחר את הביטוי החסר מבין ארבע האפשרויות הנתונות, והקף בעיגול את הספרה המייצגת אותו בדף התשובות שבנספח א'.

```
void build(int G[][V])
{
 int i, j;
 list lst;
 for(i=0; i<V; i++)
       _________(1)_________;
 for(i=0; i<V; i++)
      for(j=0; j<V; j++)
      {
          if(_________(2)_________)
          {
               lst = malloc(sizeof(listRec));
               lst->num = _________(3)_________;
               lst->next = _________(4)_________;
               vertices[i] = lst;
          }
      }
}
```

ט. הביטוי החסר (1) הוא:

1. `G[i] = NULL`
2. `vertices[i] = NULL`
3. `vertices[i] = G[i][0]`
4. `G[0][i] = 0`

המשך בעמוד 47

י. הביטוי החסר (2) הוא:

1. `vertices[i]`
2. `vertices[j]`
3. `G[i][j]`
4. `G[j][i]`

י"א. הביטוי החסר (3) הוא:

1. `lst`
2. `j`
3. `i`
4. `v`

י"ב. הביטוי החסר (4) הוא:

1. `lst`
2. `NULL`
3. `G[i]`
4. `vertices[i]`

בהצלחה!

מקום למדבקת נבחן

נספח א': דף תשובות לשאלות 3 ו־4,
לשאלון 714911, אביב תש"ף

הקף בעיגול את הספרה המייצגת את התשובה הנכונה לכל סעיף.

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

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

נספח ב': מילון מונחים (2 עמודים)

לשאלון 714911, אביב תש"ף

המונח	תרגום המונח		
	ערבית	רוסית	אנגלית
אחזור	استرجاع	Возврат, извлечение	retrieval
איבר	مُتَغيِّر / عضو	Элемент	item
אקראי	عشوائي	Случайный	random
אתחול	قيمة بدائيّة	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	مُتَغيِّر عامّ	Глобальная переменная	global variable
סדרה	سلسلة	Последовательность	series
סיבוכיות	تعقيد	Сложность (вычислений)	complexity
עדיפות	أولويّة	Приоритет	preference
עץ חיפוש בינארי מאוזן	شجرة بحث ثنائيّ متوازنة	сбалансированное двоичное дерево поиска	balanced binary search tree
ערך מוחלט	قيمة مُطْلَقة	модуль	absolute value
ערימה	كومة	Куча	heap
ערימה בינארית	كومة ثنائيّة	Двоичная куча	binary heap
פונקצייה רקורסיבית	دالّة أو عمليّة تراجعيّة	Рекурсивная функция	recursive function
צומת	مُفْتَرَق	Узел	node

המשך בעמוד 2

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