# מבנה נתונים חזרה למבחן

יוני 2020

# הוכחות

א- הוכיחו או הפריכו לפי הגדרה פורמלית (אין להשתמש ב-lim):

$3n + n^3 = \Theta(n^3)$

**תשובה:**

$c_1 * n^3 <= 3n + n^3 <= c_2 * n^3$

for all n>1 : $3n + n^3 <= 3n^3 + n^3$

and also : $n^3 <= 3n + n^3$

choose: $c_1 = 1$ $c_2 = 4$

$n_0 = 11$

$n^3 <= 3n + n^3 <= 4n^3$

## א- הוכיחו או הפריכו $5n^2 + n^3 = \Omega(n^4)$

נניח בשלילה כי הטענה נכונה על פי ההגדרה קיימים c ו – n0 עבורם לכל n > n0 מתקיים

$$5n^2 + n^3 >= cn^4$$

Divide by $n^4$

$$5 / n^2 + 1/n >= c$$

$$\lim 5 / n^2 + 1/n = 0$$

ולכן לא קיים קבוע c שהוא קטן מ- $5 / n^2 + 1/n$ בסתירה להנחה ולכן הטענה איננה נכונה.

3. הוכיחו (ע"פ ההגדרה): $3n^2+0.5n^4-2\lg n=\Omega(n^4)$

<u>הוכחה:</u>

צ"ל שקיימים קבועים חיוביים c ו- $n_0$ כך שלכל $n\geq n_0$ מתקיים:

$$3n^2+0.5n^4-2\lg n\geq cn^4$$

$$3n^2+0.5n^4-2\lg n\geq 0.5n^4-2\lg n\geq 0.5n^4-2n$$

מספיק- אם כן – להוכיח: $0.5n^4-2n\geq cn^4$ לכל $n\geq n_0$.

נעביר אגף: צ"ל: $0.5n^4\geq cn^4+2n$

צ"ל: $n^4\geq 2cn^4+4n$

צ"ל: $n^3\geq 2cn^3+4$

נבחר $c=1/4$

צ"ל: $n^3\geq n^3/2+4$

צ"ל: $n^3/2\geq 4$

צ"ל: $n^3\geq 8$

נבחר $n_0=2$ ואי השוויון שהגענו אליו אכן מתקיים לכל $n\geq n_0$!

מש"ל.

- א. הוכח או הפרך: $3n^3+7n-2$ / $5n^2-7n= \theta(8n)$
- $\lim_{n\to\infty}$ $3n$^3 + $7n$ − 2/$5n$^2 − $7n/8n$ = 3 40 = $constan$
- אם = 0 חסם עליון
- אם שואף לאינסוף חסם תחתון

- א. הוכח או הפרך:
- $f1(n) = O(f2(n))$ and $g1(n) = O(g2(n)) \Rightarrow f1(n) + g1(n) = O(f2(n) + g2(n))$
- נכון. יהיו $c1$, $n1$ הקבועים המובטחים ע"י הגדרת $f1(n) = O(f2(n))$

ויהיו $c2$, $n2$ הקבועים המובטחים ע"י הגדרת $g1(n) = O(g2(n))$

נסמן $n0 = \max\{n1, n2\}$ ו - $c = \max\{c1, c2\}$.

לכל $n \geq n0$ מתקיים:

$f1(n) + g1(n) <= cf2(n) + cg2(n))$

$f1(n) + g1(n) <= c(f2(n) + g2(n)))$

$f1(n) + g1(n) = O(f2(n) + g2(n))$

א. הוכח או הפרך: $2^{n} = \Theta(2^{n+\log n})$

לא נכון.

נניח בשלילה שנכון אז קיים קבוע חיובי c כך שלכל $n \geq n_0$ מתקיים $2^{n} \geq c\,(2^{n+\log n})$

מכאן נובע: $n = 2^{logn} \leq \frac{1}{c}$. נבחר $n$ גדול מ$n_0$ וגדול מ $1/c$ ונגיע לסתירה.

ב. הוכח או הפרך: קיימת פונקציה חיובית עולה f(n) כך ש- $f(n) = \Theta(f(n^2))$

נכון. לדוגמא: f(n)=logn $\Leftarrow$

$f(n^2)=\log(n^2)=2\log n= \Theta(\log n)$

ג. הוכח על פי הגדרה: $4n^3 + n\log n = \Theta(2n^3 + n^2)$

צ.ל. קיימים קבועים חיוביים $c_0,c_1,n_0$ כך שלכל $n \geq n_0$ מתקיים
$c_1(2n^3+n^2) \leq 4n^3+n\log n \leq c_2(2n^3+n^2)$.

לגבי האי שוויון הימני, לכל $n \geq 1$ מתקיים: $4n^3+n\log n<4n^3+n^2<5n^3<3(2n^3)<3(2n^3+n^2)$

ולכן נבחר $c_2=3$.

לגבי האי שוויון השמאלי נבחר $c_1=1$, ונקבל שלכל $n \geq 1$ מתקיים:

$2n^3+n^2<2n^3+n^3<4n^3<4n^3+n\log n$

בהינתן n מספרים שלמים בטווח [1..k], הציעו אלגוריתם המעבד מראש את הקלט (**preprocess**) בזמן $O(n+k)$, ולאחר מכן בזמן $O(1)$ מחזיר כמה מספרים נמצאים בטווח [a..b] עבור כל a ו-b נתונים.

```
Range(A, k, a, b)
  Preprocess :
    for i ← 1 to k
      C[i] ← = 0
    for j ← 1 to n
      C[A[j]] ← C[A[j]] + 1  // C[i] = the number of appearances of i in A.
    for i ← 2 to k
      C[i] ← C[i] + C[i-1]   // C[i] = the number of elements in A that are ≤ i
  return (C[b] - C[a-1])
```

נתונות n מילים באנגלית (ניתן להניח שכולם ב-lowercase). נתון שאורך המילים הוא קבוע. כיצד ניתן למיין את המילים בסדר לקסיקוגרפי בזמן O(n)?

- יהי m אורך המילה המקסימלית בקלט. (m קבוע).
- נייצג כל אות כספרה בבסיס 27, כלומר בטווח ( 0-26) , כלומר $a=1,b=2,\dots z=26$. (O(n)).
- מילים שמספר אותיותיהן $k<m$ יושלמו בסוף באפסים. (O(n))
- נבצע מיון בסיס עם $d=m$ ו- $b=27$ .
  זמן ריצה: $O(d(n+b))=O(n)$
- נתרגם כל מספר חזרה לאותיות $O(n)$.

**<u>Example:</u>**

Lexicographical-sort input: {blue, red, green}
Radix-sort input: { (2,12,21,5,0), (18,5,4,0,0), (7, 18,5,5,14) }
Radix-sort output: { (2,12,21,5,0), (7, 18,5,5,14) , (18,5,4,0,0) }
Lexicographical-sort output: {blue, green, red}

נתונה קבוצת מספרים שלמים בטווח $[1..n^3]$ . הציעו אלגוריתם יעיל למיונם.

- מיון מבוסס השוואות ייקח $O(nlogn)$.
- מיון מניה:

  $k=n^3 \rightarrow O(n+n^3)=O(n^3)$
- מיון בסיס:

  $b=10,\ d=\lceil 3\log_{10}n \rceil,\ \rightarrow O(\log_{10}n\ (n+10)=O(n\log_{10}n)$

- נשתמש במיון בסיס לאחר עיבוד ראשוני:

  ראשית נתרגם את כל המספרים למספרים בבסיס n, בזמן $O(n)$.

  עתה נריץ את מיון בסיס.

  כיון שכל המספרים הם בטווח $[1..\ n^3]$ נקבל לכל היותר 4 ספרות בבסיס n.

  זמן הריצה לאלגוריתם המוצע יהיה איפוא: $d=4,b=n \rightarrow O(4(n+n)) =$

  $O(n)$.

5. כיצד ניתן לממש מחסנית בעזרת תור קדימויות?

<u>תשובה:</u>

מבנה הנתונים יכלול מונה, שעולה ב-1 על כל איבר חדש שנכנס למחסנית.

לכל איבר יהיה שדה נוסף מסוג int, נניח בשם: t.

את אברי המחסנית נשמור במבנה מסוג: תור קדימויות, שיפעל על פי ערכי: t.

פעולת Push(S,x) תעשה כך: נוסיף 1 למונה, נציב את ערכו בשדה t של x, ונוסיף את x לתור הקדימויות ע"י הפעלת הפונקציה: Insert (S,x) .

פעולת Pop(S) תעשה ע"י הפעלת הפונקציה: Extract-Max(S) מתור הקדימויות.

האיבר שיצא הוא האיבר עם הערך המקסימאלי, כלומר זה שנכנס אחרון.

בהינתן עץ AVL כלשהו, האם ניתן לבנות עץ זהה ע״י סדרת פעולות רגילות של הכנסת איבר לעץ בינארי (ללא סיבובים)?

תשובה: כן. נכניס את אברי העץ לפי רמות (משמאל לימין). קודם את השורש, אח״כ את בניו לפי סדר, אח״כ את נכדיו וכו׳.

נתון עץ חיפוש בינארי מאוזן מסוג AVL. כתבו פונקציה שמקבלת קודקוד p בעץ ומבצעת עליו גלגול LL. ( p הוא הקודקוד שבו הופר האיזון. זהו השורש של תת העץ שעליו יש לבצע את הגילגול). על הפונקציה להחזיר מצביע לשורש החדש של תת העץ המעודכן (p או קודקוד אחר).

```
LL(p)
{
        B = p;
        A = left(B);
        AR = right(A);
        Parent(A) = parent(B);
        Right(A) = B;
        Parent(B) = A;
        Left(B) = AR;
        Parent(AR) = B;
        Return(A);
}
```

1. נתון עץ **חיפוש** בינארי המכיל מספרים בין 1 ל- 1000 (לאו דוקא את כולם). אנו מעוניינים לחפש את המספר 363. האם הסידרה הבאה יכולה להיות סדרת המספרים בהם ביקרנו במהלך החיפוש? (משמאל לימין):

935, 278, 347, 621, 299, 392, 358, 363

<u>תשובה</u>: לא יתכן. 299 היה אמור להיות בתת-העץ השמאלי של- 347 ולא בתת העץ הימני שלו.

2. מהו העץ הבינארי (לאו דוקא עץ חיפוש) שסריקותיו הן (משמאל לימין) :

סדר תחילי (Pre-order) : 9,6,2,10,4,7,8,5,3,1,0

סדר תוכי (In-order) : 10,2,4,6,9,8,7,3,5,1,0

תשובה : אילו היה מדובר בעץ **חיפוש** בינארי, הסידרה שהתקבלה על פי in-ordeer היתה צריכה להיות ממוינת. לכן נתיחס לסידרה זו כסדר מסוים בין המספרים, ועל פיו ועל פי הסידרה הראשונה ניתן בקלות לבנות את העץ.

השורש הוא 9 כי הוא הראשון על פי Pre-order. ממנו עוברים ל-6. מכיון שבסידרה השניה 6 מופיע לפני 9, סימן ש-6 הוא בנו השמאלי של 9. ממנו עוברים ל-2. מכיון ש- 2 נמצא לפני 6 בסידרה השניה סימן שהוא בנו השמאלי של 6, וכן הלאה.

4. כתבו פונקציה המקבלת עץ בינארי ומחזירה את גובה העץ.

מהי סיבוכיות זמן הריצה של הפונקציה ? נמקו.

<u>תשובה</u>:

```
height(x)
{
        If x==null return (0);
        If (left(x)==null && right(x)==null) return (0);
        Return (max(height (left(x)), height (right(x))) + 1);
}
```

זמן הריצה : $\Theta(n)$. בדומה ל-in-order, הפונקציה מבקרת בכל קוד' בעץ מס' קבוע של פעמים. יש כאן ביצוע של מס' קבוע של פעולות+ קריאה רקורסיבית אחת של הפונ' לצד שמאל של העץ + קריאה רקורסיבית אחת לצד ימין של העץ. בסה"כ סריקה אחת של כל העץ.

5. כתבו פונקציה המקבלת עץ בינארי ומחזירה את מספר הבנים היחידים בעץ.

מהי סיבוכיות זמן הריצה של הפונקציה? נמקו.

<u>תשובה</u>:

```
Single-sons (x)
{
        If x==null return(0);
        If (left(x)==null && right(x)==null) return(0);
        If (left(x)==null && right(x)!=null) return(Single-sons(right(x)) +1);
        If (left(x)!=null && right(x)==null) return(Single-sons(left(x)) +1);
        Return(Single-sons(left(x)) + Single-sons(right(x)));
```

זמן הריצה: $\Theta(n)$. גם כאן יש סריקה אחת של כל העץ.

6. כתבו פונקציה המקבלת עץ בינארי ומחזירה את סכום ערכי הבנים הימניים בעץ.

מהי סיבוכיות זמן הריצה של הפונקציה? נמקו.

<u>תשובה</u>:

```
Sum-right-sons (x)
{
        If x==null return(0);
        If (right(x)==null) return(Sum-right-sons(left(x));
    Return(key(right(x)) + Sum-right-sons(left(x)) + Sum-right-sons(right(x)));
}
```

זמן הריצה: $\Theta(n)$. גם כאן יש סריקה אחת של כל העץ.

1. הוכיחו או סיתרו: "ניתן לממש תור ע"י רשימה מקושרת חד-כיוונית, כך שכל הפעולות הבסיסיות על התור יתבצעו בזמן $O(1)$".

תשובה: **נכון**. נשתמש ברשימה חד-כיוונית עם מצביע לסוף הרשימה – tail.

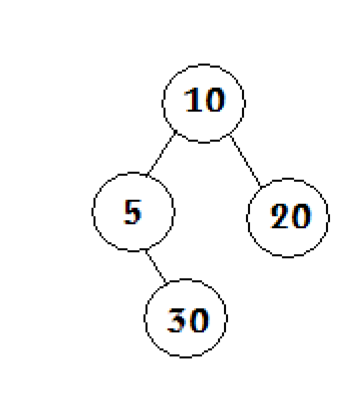
**הכנסת איבר** לתור תעשה ע"י הוספת איבר לסוף הרשימה (לאחר ה- tail), וקידום ה- tail.

**הוצאת איבר** תעשה ע"י הוצאת האיבר הנמצא בראש הרשימה,כלומר ה- head וקידומו.

**בדיקת ריקנות** תעשה ע"י בדיקה האם head==Null.

2. הוכיחו או סיתרו: " עץ בינארי שבו בכל צומת נמצא ערך הגדול מהערך בבנו השמאלי וקטן מהערך בבנו הימני הוא בעצם **עץ חיפוש בינארי**".

<u>תשובה:</u> **לא נכון.** דוגמה נגדית:



- נכון/לא נכון

א- יש למיין $n$ מספרים טבעיים בתחום בין 1 ל- $2^n$ . מיון מנייה – counting sort יהיה יותר יעילה ממיון ערמה – heap sort.

לא נכון – במיון מנייה הסיבוכיות יהיה $O(n + 2^n) <== O(2^n)$

לעומת מיון ערמה הסיבוכיות של המיון של הערכים יהיה $O(n \log n)$

- נכון/לא נכון

א- עומקו של כל עץ חיפוש בינארי – binary search tree המכיל n איברים הינו $O(\log n)$

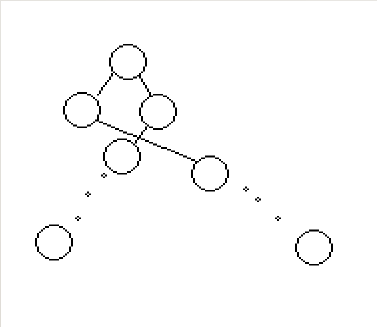
לא נכון – עץ מנוון

1. מהו זמן הריצה של מחיקת איבר מעץ חיפוש בינארי, במקרה הטוב, הגרוע והממוצע? נמקו.

בהנחה שנתון מצביע ישירות אל האיבר שאותו צריך למחוק, ומכל איבר יש מצביע אל האבא:

**במקרה הטוב:** $O(1)$. לדוגמה כאשר מוחקים עלה או קודקוד עם בן אחד.

**במקרה הגרוע:** $\Theta(n)$. לדוגמה כאשר רוצים למחוק את השורש בעץ הבא:



יש להחליף את השורש עם העוקב (או עם הקודם), ולכך דרושים בדוגמה זו כ- n/2 צעדים.

**במקרה הממוצע:** $O(\lg n)$ – במקרה זה העצים מאוזנים, ולכן אורך שביל ממוצע הינו $\Theta(\lg n)$, וזהו מקסימום הזמן שידרש להגיע לעוקב – אם צריך.

1. כתבו ונמקו מהו סדר-הגודל של זמן הריצה של כל אחת הפונקציות הבאות (כפונקציה של n):

```
fun_a              fun_b                          fun_c
i = 1              for (i=n; i = i/n; i < 1)             for (i=1; i =
2*i; i < n*n)
while (i < n)      { …}                              {…}
{ …
   …
   i = i * 4
}
```

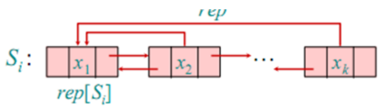
fun_a - שינוי של פרמטר הלולאה ע"י הכפלת קבוע בעצמו מביאה להתנהגות של log ובמקרה דנן ל- $O(\log_4 n)$

fun_b - פרמטר הלולאה מתחיל ב- n וכבר אחרי שלב אחד (n/n) הוא מגיע ל- 1 כך שבשלב השני יעצור כלומר $O(1)$

fun_c - הפרמטר רץ עד $n^2$ ושינוי של פרמטר הלולאה ע"י הכפלת 2 בעצמו מביאה כאמור להתנהגות של log ולכן סה"כ $O(\log_2 n^2)$ שזה $O(\log_2 n)$

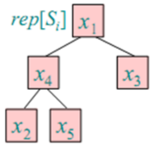
## ציינו 3 שיטות שונות לייצוג קבוצות זרות, ותארו בכמה משפטים כל שיטה

א. כל קבוצה שמורה בתוך רשימה מקושרת אחת, כאשר ראש רשימה מייצג קבוצה, וכל איבר מצביע לראש הרשימה

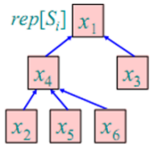



ב. כל קבוצה שמורה בתוך עץ בינארי מאוזן, השורש של כל עץ בינארי מייצג את הקבוצה.

$S_i = \{x_1, x_2, x_3, x_4, x_5\}$




ג. כל קבוצה שמורה בתוך עץ לא בינארי, כאשר כל קודקוד מצביע על אביו, והשורש של כל עץ מייצג את הקבוצה.



## מבין השיטות שציינתם, מה השיטה/השיטות שהיינו מעדיפים אם חשוב לנו שפעולת ה- Find תהיה יעילה ככל שאפשר? הסבירו.

אם רוצים שפעולת ה- Find תהיה מהירה ביותר, נעדיף את היצוג של רשימה מקושרת שבה לכל קודקוד יש מצביע לראש הרשימה, כיון שאז המציאה מיהו ראש הרשימה לוקחת $O(1)$.

ג.מבין השיטות שציינתם, מה השיטה/השיטות שהיינו מעדיפים אם חשוב לנו שפעולת ה- Union תהיה יעילה ככל שאפשר? הסבירו.

במקרה זה, נבחר את שיטת העצים ההפוכים (כל קודקוד מצביע לאביו), שבה ביצוע ה- union הוא בסדר גודל של $O(1)$ (פשוט נותנים לשורש העץ הקטן אבא חדש שהוא שורש העץ הגדול יותר).

נתון עץ B מדרגת מקסימום m. כתבו אלגוריתם שמקבל את שורש העץ הנ"ל ומחזיר כמה מפתחות יש בו.
מה סיבוכיות האלגוריתם שכתבתם?

```
Int count(BNode *p)
If p==NULL return 0
If p->son[0]==NULL //עלה
   Return p->numKeys
//else
Sum=p->numKeys
For (i=0; i<numKeys+1; i++)
     Sum+=count(p->son[i])
Return sum
```

סיבוכיות: theta(n) כאשר n הוא מספר המפתחות בעץ: סריקת העץ כולו בסדר preorder.

Given a hash table with $m=11$ entries and the following hash function $h_1$ and step function $h_2$:

$h_1(key) = key \bmod m$
$h_2(key) = \{key \bmod (m-1)\} + 1$

Insert the keys {22, 1, 13, 11, 24, 33, 18, 42, 31} in the given order (from left to right) to
the hash table using each of the following hash methods:

a. Chaining with $h_1 \Rightarrow h(k) = h_1(k)$
b. Linear-Probing with $h_1 \Rightarrow h(k,i) = (h_1(k)+i) \bmod m$
c. Double-Hashing with $h_1$ as the hash function and $h_2$ as the step function $\Rightarrow h(k,i) = (h_1(k) + ih_2(k)) \bmod m$

| | Chaining | Linear Probing | Double Hashing |
|---|---|---|---|
| 0 | 33 → 11→ 22 | 22 | 22 |
| 1 | 1 | 1 | 1 |
| 2 | 24 → 13 | 13 | 13 |
| 3 | | 11 | |
| 4 | | 24 | 11 |
| 5 | | 33 | 18 |
| 6 | | | 31 |
| 7 | 18 | 18 | 24 |
| 8 | | | 33 |
| 9 | 31→ 42 | 42 | 42 |
| 10 | | 31 | |

2. נתונה טבלת ערבול שבה התנגשויות נפתרות בשיטת Open Addressing.
הכניסו לטבלה את המפתחות הבאים: 10, 22, 31, 4, 15, 28, 17, 88, 59 (מימין לשמאל), כאשר גודל הטבלה הוא $m = 11$, תוך שימוש בשיטות הבאות:

א. **Linear Probing:** $h(k,i) = (h_1(k) + i) \bmod m$, כאשר $h_1(k) = k \bmod m$.

ב. **Quadratic Probing:** $h(k,i) = (h_1(k) + c_1 i + c_2 i^2) \bmod m$
כאשר $h_1(k)$ כמו בסעיף א' ו- $c_1 = 1$ $c_2 = 3$.

ג. **Double Hashing:** $h(k,i) = (h_1(k) + ih_2(k)) \bmod m$
כאשר $h_1(k)$ כמו בסעיף א', ו- $h_2(k) = 1+(k \bmod (m-1))$.

ציירו את הטבלה המתקבלת בכל אחד מהמקרים.

שאלה 2:

נתונה טבלת ערבול שבה התנגשויות נפתרות בשיטת Open Addressing.
נכניס לטבלה את המפתחות הבאים: 10, 22, 31, 4, 15, 28, 17, 88, 59 (מימין לשמאל), כאשר גודל הטבלה הוא $m = 11$, תוך שימוש בשיטות הבאות:

א. **Linear Probing**: $h(k,i) = (h_1(k) + i) \bmod m$, כאשר $h_1(k) = k \bmod m$.

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| 22 | 88 | | | 4 | 15 | 28 | 17 | 59 | 31 | 10 |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |

ב. **Quadratic Probing**: $h(k,i) = (h_1(k) + c_1 i + c_2 i^2) \bmod m$
כאשר $h_1(k)$ כמו בסעיף א' ו- $c_1 = 1$ $c_2 = 3$.

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| 22 | | 88 | 17 | 4 | | 28 | 59 | 15 | 31 | 10 |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |

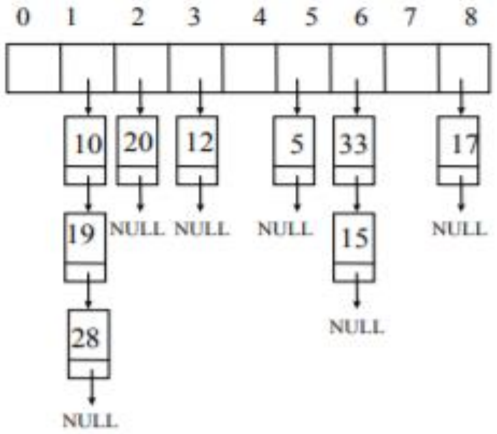
שימו לב שלא כל סדרות החיפוש הן פרמוטציה (תמורה) של כל המקומות בטבלה, וזה כמובן אינו טוב.
לדוגמה: במקרה של המפתח 88.

ג. **Double Hashing**: $h(k,i) = (h_1(k) + ih_2(k)) \bmod m$
כאשר $h_1(k)$ כמו בסעיף א', ו- $h_2(k) = 1+(k \bmod (m-1))$.

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| 22 | | 59 | 17 | 4 | 15 | 28 | 88 | | 31 | 10 |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |

שאלה 3:

נתונה טבלת ערבול שבה התנגשויות נפתרות ע"י שרשור, ופונקציית הערבול: $h(k) = k \bmod m$.
נכניס את המפתחות 5, 28, 19, 15, 20, 33, 12, 17, 10 (מימין לשמאל) לטבלה בגודל $m = 9$.



הראו איך ניתן למיין אוסף מחרוזות בסדר לקסיקוגרפי תוך שימוש ב- Trie. נתחו את סיבוכיות המיון



## מיון מחרוזות באמצעות Trie

ניתן להשתמש בעובדה שלמחרוזות יש מבנה - שרשרת תווים מעל אלף-בית מאורך קבוע $\Sigma$ - כדי למיינן מהר יותר מאשר ע"י השוואות של מחרוזות. נשתמש ב- Trie באופן הבא:

**האלגוריתם**

1. הכנס את $S_1, \dots, S_n$ ל- Trie.
2. עבור על ה- trie לפי סדר preorder וכתוב לפלט את המסלול לכל עלה. (המסלול נמצא במחסנית הרקורסיה).

דוגמא: נתונות המחרוזות $ac, a, bc$, כאשר תו סיום-מחרוזת הוא $ (הקטן ביותר לקסיקוגרפית). המחרוזות הממוינות הן $a\$, ac\$, bc\$$.