מבני נתונים
תרגיל 7 - פתרונות

1. הכניסו את האברים הבאים לטבלאת גיבוב בגודל 11:

$$10, 22, 31, 4, 15, 28, 17, 88, 59$$

הטבלא עובדת ללא רשימות מקושרות. נגדיר את פונקצית הגיבוב: $f(k) = k \ mod \ 11$. תארו את מצב הטבלא בסוף הכנסת האברים.

פתרון:

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

2. נכניס מחרוזות שונות לטבלאת גיבוב בגודל $m = 7$ שמשתמשת ברשימות מקושרות. נניח שזהו קוד המחשב של האותיות הבאות:

$$a = 1, b = 2, c = 3, d = 4$$

הניחו שהבסיס שאנו עובדים לפיו בחישוב פונקצית הגיבוב של מחרוזת הוא בסיס 5. למשל, פונקציות הגיבוב של המחרוזת abd היא:

$$h(abd) = (1 \cdot 5^2 + 2 \cdot 5 + 4) \ mod \ 7 = 39 \ mod \ 7 = 4$$

הכניסו את המחרוזות הבאות לטבלא:

$$aaa, ba, bbdd, c, aba, dcdb$$

פתרון:

0	
1	aba
2	$dcdb$
3	$c, bbdd, aaa$
4	
5	
6	ba

3. חישבו איך לחשב את פונקצית הגיבוב של מחרוזות ארוכות במיוחד. תנו אלגוריתם לינארי שתמיד שומר על המספרים לא גדולים בהרבה מ־m:

```
void hash(String s, int r, int m);
```

מה למשל יהיה הערך של מחרוזת של עשרה a רצופים בפרמטרים של השאלה הקודמת?

פתרון: אלגוריתם אפשרי הוא כזה:

```
void hash(String s, int r, int m)
{
  int val = 0;
  int rInPow = 1;
  for (int i=s.length-1; i>=0 ; i--) {
    val += ascii(s.charAt(i)) * rInpow
    val = val \mod m
    rInPow *= r
    rInPow = rInPow \mod m
  }
}
```

ואפשר טיפה יותר אלגנטי עם מחשבה מתמטית:

```
void hash(String s, int r, int m)
{
  int val = 0;
  for (int i=0; i<s.length; i++) {
    val *= r
    val += ascii(s.charAt(i))
    val = val \mod m
  }
}
```

כדי לחשב את הערך של המחרוזת הספציפית, נחשוב יותר על האלגוריתם הראשון (אבל בסדר קצת אחר) ונחשב את הסדרה $1, 5, 5^2, 5^3, ..., 5^9$ כולה במודול 7.

נחשב אותם אחד אחד כך שאף פעם המספרים לא יגדלו.

1	$1 \ mod \ 7 = 1$
5	$5 \ mod \ 7 = 5$
5^2	$25 \ mod \ 7 = 4$
5^3	$5 * 4 \ mod \ 7 = 20 \ mod \ 7 = 6$
5^4	$5 * 6 \ mod \ 7 = 30 \ mod \ 7 = 2$
5^5	$5 * 2 \ mod \ 7 = 10 \ mod \ 7 = 3$
5^6	$5 * 3 \ mod \ 7 = 15 \ mod \ 7 = 1$
5^7	$5 * 1 \ mod \ 7 = 5$
5^8	$5 * 5 \ mod \ 7 = 25 \ mod \ 7 = 4$
5^9	$5 * 4 \ mod \ 7 = 20 \ mod \ 7 = 6$

עכשיו מה שיש לנו, כיון שכל האותיות הן a זה פשוט לסכום את כולם. אם היו כל מני אותיות היה צריך להכפיל בערכים המתאימים ולסכום.

$$1 + 5 + 4 + 6 + 2 + 3 + 1 + 5 + 4 + 6 = 37$$

ואת זה נחשב כמובן מודולו 7 ונקבל 2.

4. אנו עובדים עם טבלאת גיבוב (Hash) שעובדת ללא רשימות מקושרות. בכיתה ראינו כי ניתן לתמוך בפעולת המחיקה של אבר בטבלא ע"י סימונו בסימון מיוחד כמחוק ולשנות את הפעולות שלנו כדי שידעו לעבוד איתו.

כיתבו את פעולת המחיקה בצורה שלא תשתמש בסימן מיוחד כזה ומבלי לשנות את הקוד המקורי של הכנסה וחיפוש. הנה הקוד שלהן מהשיעור. אל תשנו דבר מקוד זה, הוסיפו רק שיטה שמבצעת מחיקה.

```
private int inc(int p)
{
  if (p < T.length - 1)
    return p+1;
  else
    return 0;
}
private int findIndex(int key)
{
  int p = h(key)
  while (T[p] != null && T[p].getKey() != key)
    p = inc(p);
  return p;
}
```

```
void insert(Element e) {
  T[findIndex(e.getKey())] = e;
}
Element search(int key) {
  return T[findIndex(key)];
}
```

פתרון: הרעיון הוא למצוא את המקום שאותו אנו רוצים למחוק ולמחוק אותו. הבעייה שאלו שאחריו יכול להיות שכבר לא במקום הנכון. לכן צריך ללכת אחריו כל עוד יש אברים - ברגע שנגיע למקום ריק זה אומר שאלו אחרי זה כבר לא קשורים. את כל האברים האלה, נמחוק מהטבלא ונכניס חזרה. היינו יכולים למחוק את כולם ואז להכניס אותם אחד אחד חזרה, אבל למעשה אין צורך. אפשר לטפל בהם אחד אחד, כי ברגע שמחקנו אחד, ואז נכניס אותו חזרה, הכי רחוק שהוא יגיע זה המקום הישן שלו. כלומר אין צורך למחוק את אלו שאחריו בינתיים.

```
void delete(Element e) {
  int p = findIndex(e.getKey())

  if (T[p] == null)
    return; // Not found
  T[p] = null;
  p = inc(p);
  while(T[p] != null) {
    Element temp = T[p];
    T[p] = null;
    insert(temp);
    p = inc(p);
  }
}
```

5. עבדו לפי האלגוריתם הסופי (כפי שמופיע בדף הנוסחאות) שלמדנו של Union-Find, והדגימו את הפעולות הבאות בתרשימים:

```
makeSet(a);
makeSet(b);
...
makeSet(k);
```

```
union(a,b);
union(a,c);
union(e,d);
union(f,g);
union(h,i);
union(j,i);

union(c,d);
union(f,h);

union(f,k);

union(a,f);
```

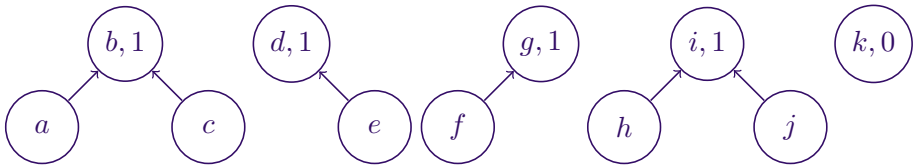
לאחר כל קבוצת פעולות יש לצייר את המצב הנוכחי. ציינו בשורש של כל עץ את ה־rank שלו. שימו לב גם שה־rank של קודקוד לאו דוקא מייצג בדיוק את הגובה שלו, אלא הוא יכול להיות קצת יותר וזה בגלל ש־find יכול לקצר מסלולים.

כדי שהתוצאות שלכם תצאנה זהות לפתרון, הקפידו שכאשר אין הבדל בין עומקי שתי הקבוצות שמאחדים, השנייה היא זאת שנהיית השורש.

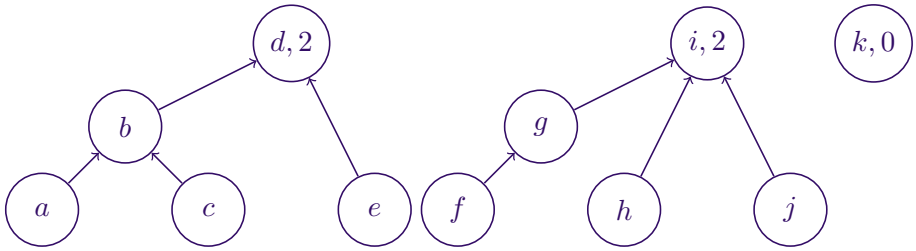
פתרון:

יצירה:

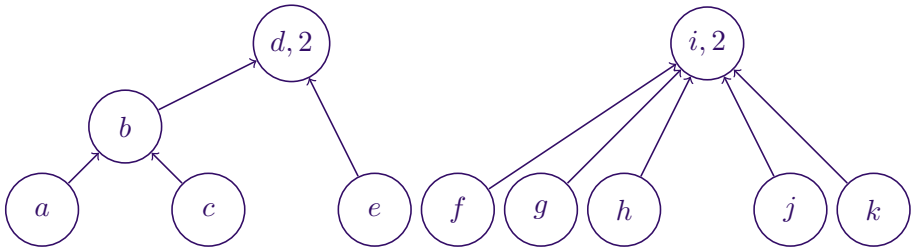
פעולות ראשונות:



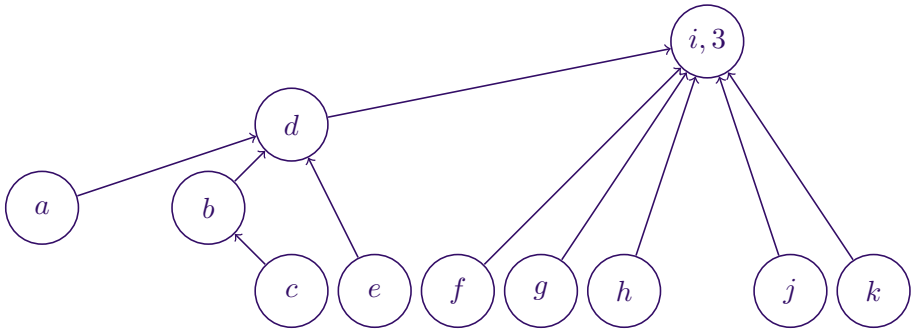
עוד שתי פעולות:



עוד אחת - שימו לב שלמרות שהעץ עכשיו עומקו קטן, ה-rank שלו לא יודע את זה.



ואחרונה:



6. בכיתה ראינו איך לתמוך בפעולות של union-find ע"י הוספת שדות למחלקה Element. במציאות, פעמים רבות לא ניתן לעשות זאת. כתבו מחלקה שיהיו לה השיטות:

```
class UnionFind {
  public UnionFind(int m) // constructor

  void makeSet(Element e)
```

```
  Element find(Element e)
  void union(Element e1, Element e2)
}
```

כאשר אין אתם יודעים דבר על המחלקה Element מלבד זה שיש לה שיטה int getKey(), שאם מפעילים אותה על אלמנט, היא מחזירה ערך יחודי שלא שייך לאף אבר אחר.

כדי לממש שיטות אלו תצטרכו להשתמש גם בטבלאת גיבוב וגם באלגוריתם שלמדנו ל union-find.

השתמשו בטבלאות גיבוב בשיטת הרשימה המקושרת. לכל אלמנט שמוסף יווצר Node, כדי שיוכל להיכנס לטבלאת הגיבוב וגם כדי שיוכל להשתתף ביער ההפוך. תכנתו את כל הקוד כולל כל השיטות שצריך.

```
class Node {
  Element e;
  Node next, parent;
  int rank;
}
```

פתרון:

```
class Node {
  Element e;
  Node next, parent;
  int rank;

  public Node(Element e, Node next) {
    this.e = e;
    this.next = next;
    parent = null;
    rank = 1;
  }
}
class UnionFind {
  Node[] T;

  public UnionFind(int m)
  {
    T = new Node[m];
```

```
}

private int h(Element e)
{
  return (e.getKey() mod m);
}

void makeSet(Element e)
{
  int p = h(e);
  T[p] = new Node(e, T[p], null);
}

Element find(Element e)
{
  Node n;
  for (n = T[h(e)]; n.e != e; n = n.next);
  Node root = n;
  while (root.parent != root)
    root = root.parent;
  while (n.parent != root) {
    Node temp = n.parent
    n.parent = root;
    n = temp;
  }
  return root;
}
void union(Element e1, Element e2)
{
  Node n1 = find(e1);
  Node n2 = find(e2);

  if (n1.rank < n2.rank)
    n1.parent = n2;
  else if (n2.rank < n1.rank)
    n2.parent = n1;
  else {
    n1.parent = n2;
    n2.rank++;
  }
```

```
    }
}
```