תרגול 10

Binary search trees , Hash tables

עצים

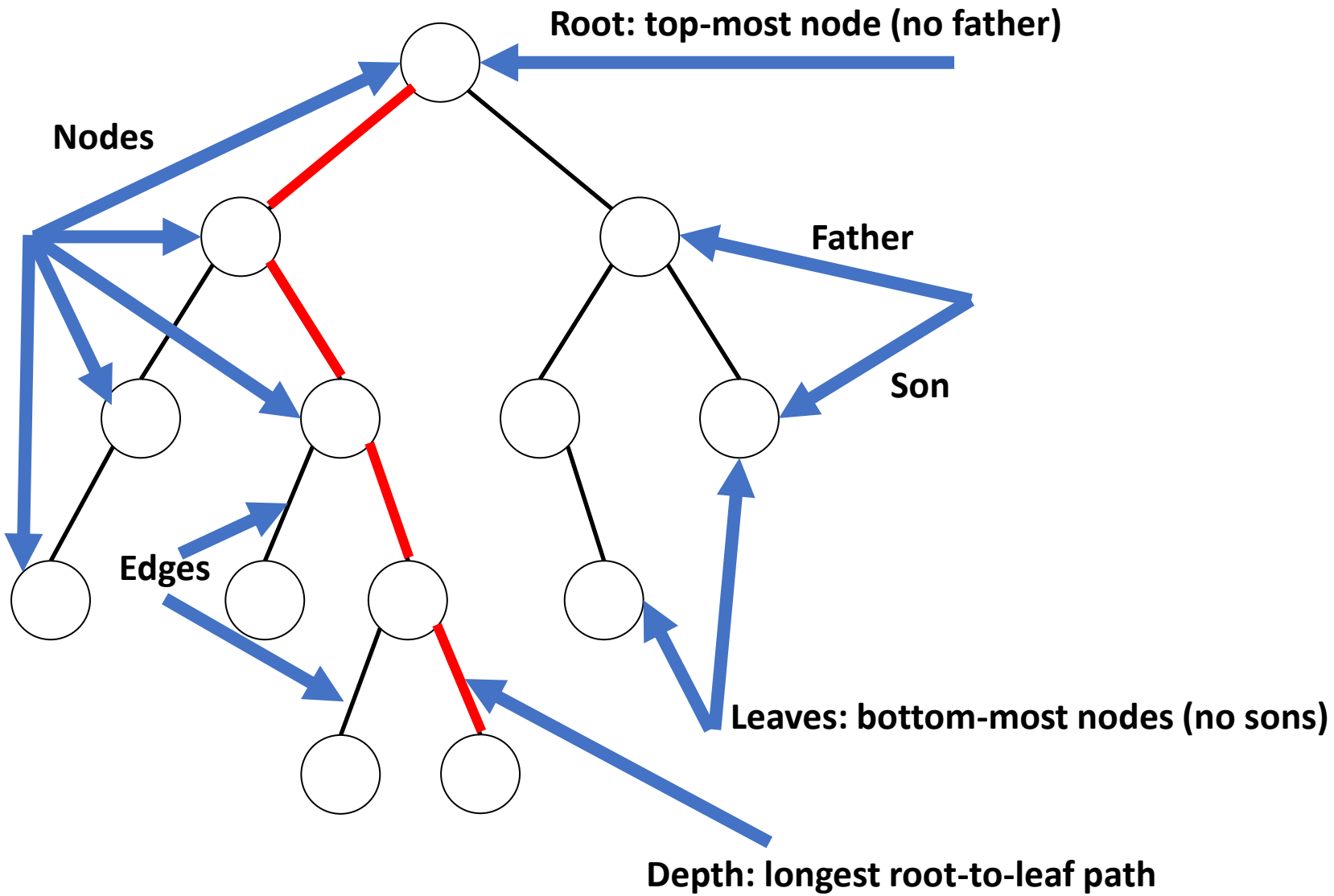


Diagram courtesy of Data Structures course (Haim Kaplan, Uri Zwick)

Binary search trees

- עצים בהם לכל צומת יש $2 \geq$ בנים (left, right – אולי None)

בנוסף:

- כל צומת מחזיק גם מפתח ייחודי (key), וערך (value)
- כל צומת v מקיים את האינווריאנטה הבאה:

$v.key$ גדול ממש מכל המפתחות בתת העץ ששורשו $v.left$

וגם

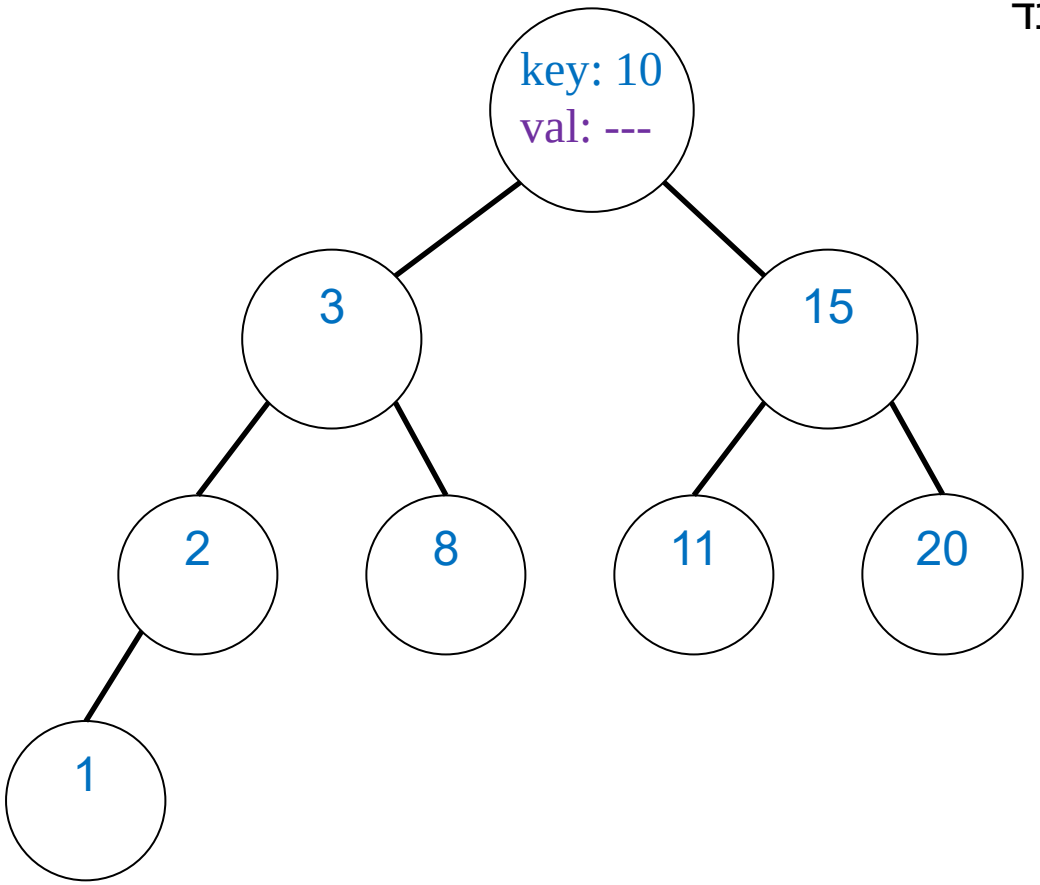
$v.key$ קטן ממש מכל המפתחות בתת העץ ששורשו $v.right$

כלומר: $keys(v.left) < v.key < keys(v.right)$

- האינווריאנטה מתייחסת למפתחות בלבד

דוגמה

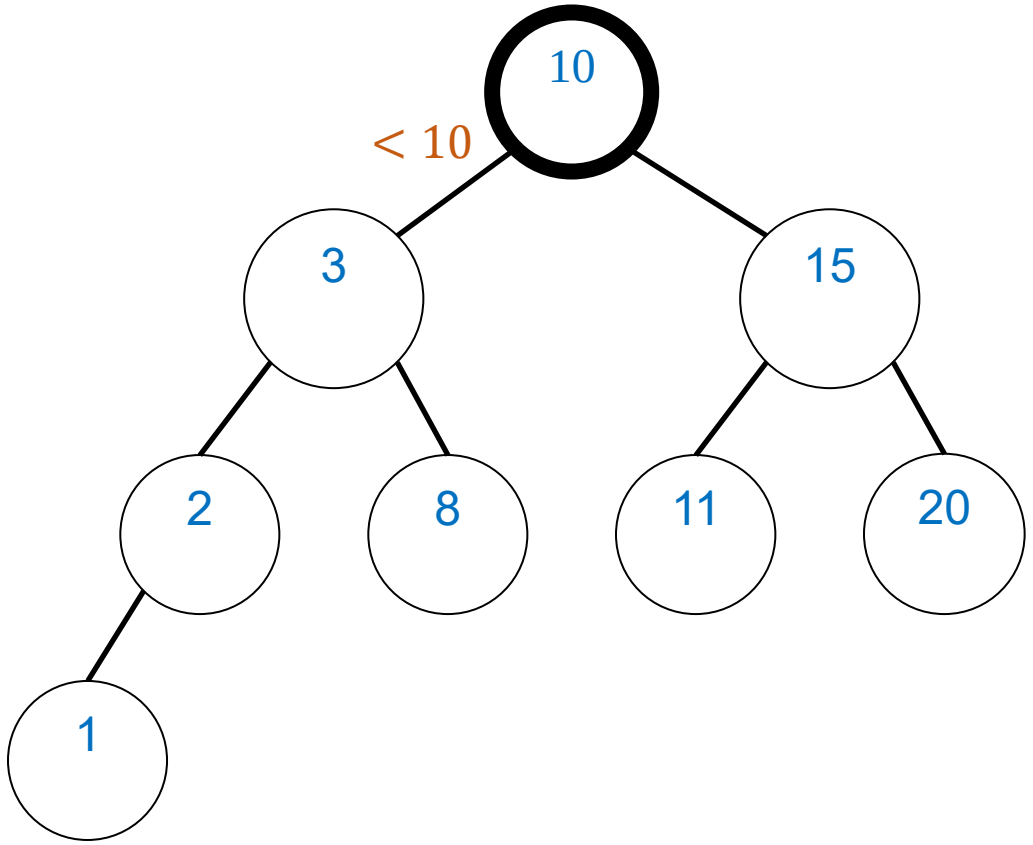
האינוריאנטה מתייחסת למפתחות בלבד



הכנסה לעץ חיפוש בינארי

נכניס לעץ את המפתח 9 – זהו מפתח שאינו קיים בעץ

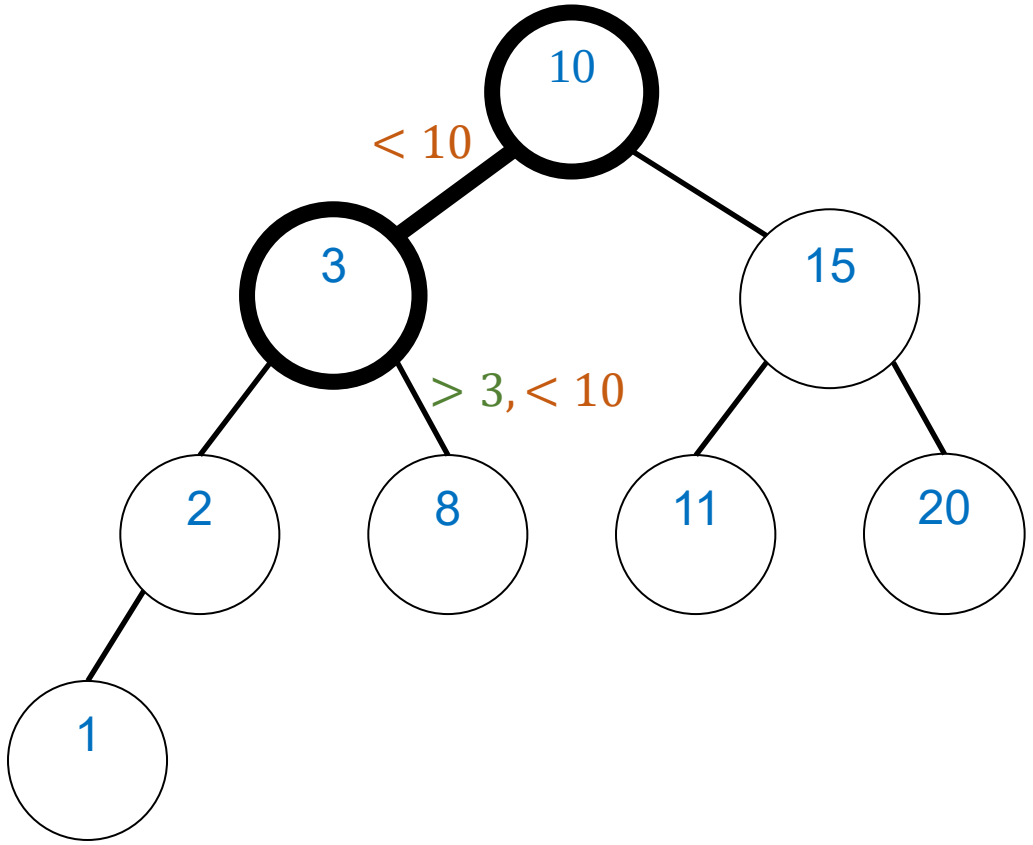
9 קטן מ 10 ולכן נמשיך לתת העץ ששורשו הבן השמאלי



הכנסה לעץ חיפוש בינארי

נכניס לעץ את המפתח 9 – זהו מפתח שאינו קיים בעץ

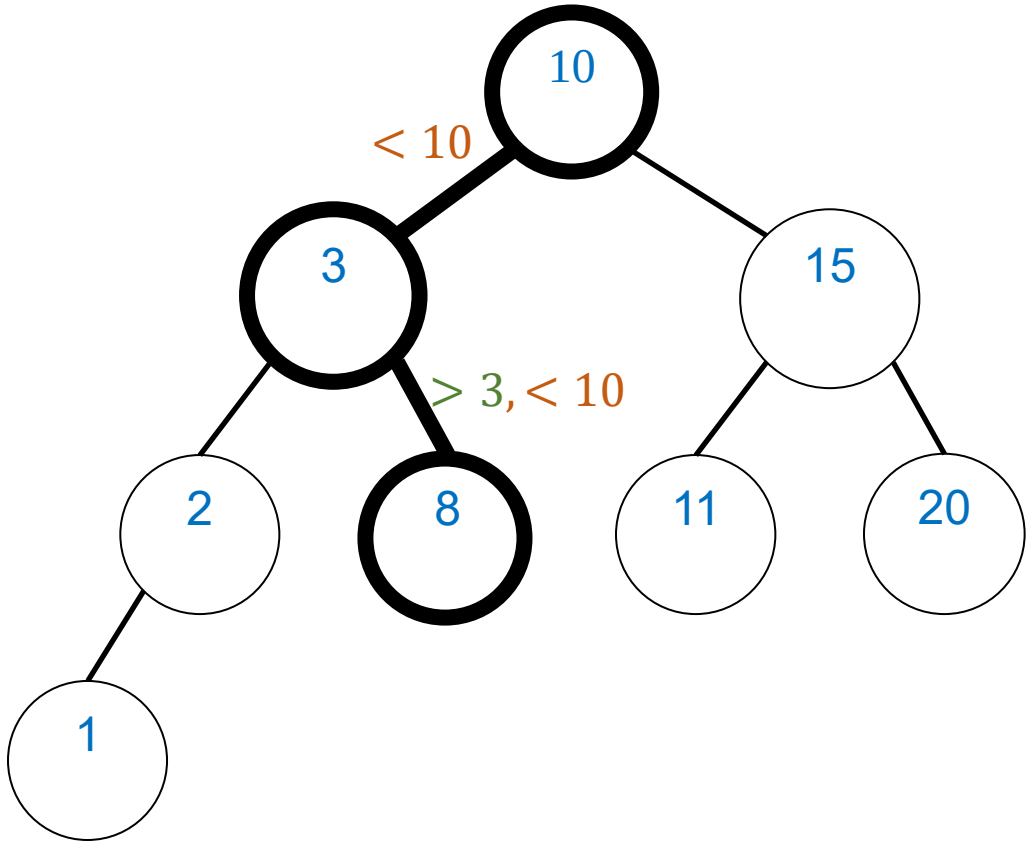
9 גדול מ 3 ולכן נמשיך לתת העץ ששורשו הבן הימני



הכנסה לעץ חיפוש בינארי

נכניס לעץ את המפתח 9 – זהו מפתח שאינו קיים בעץ

9 גדול מ 8 ולכן נמשיך לתת העץ ששורשו הבן הימני

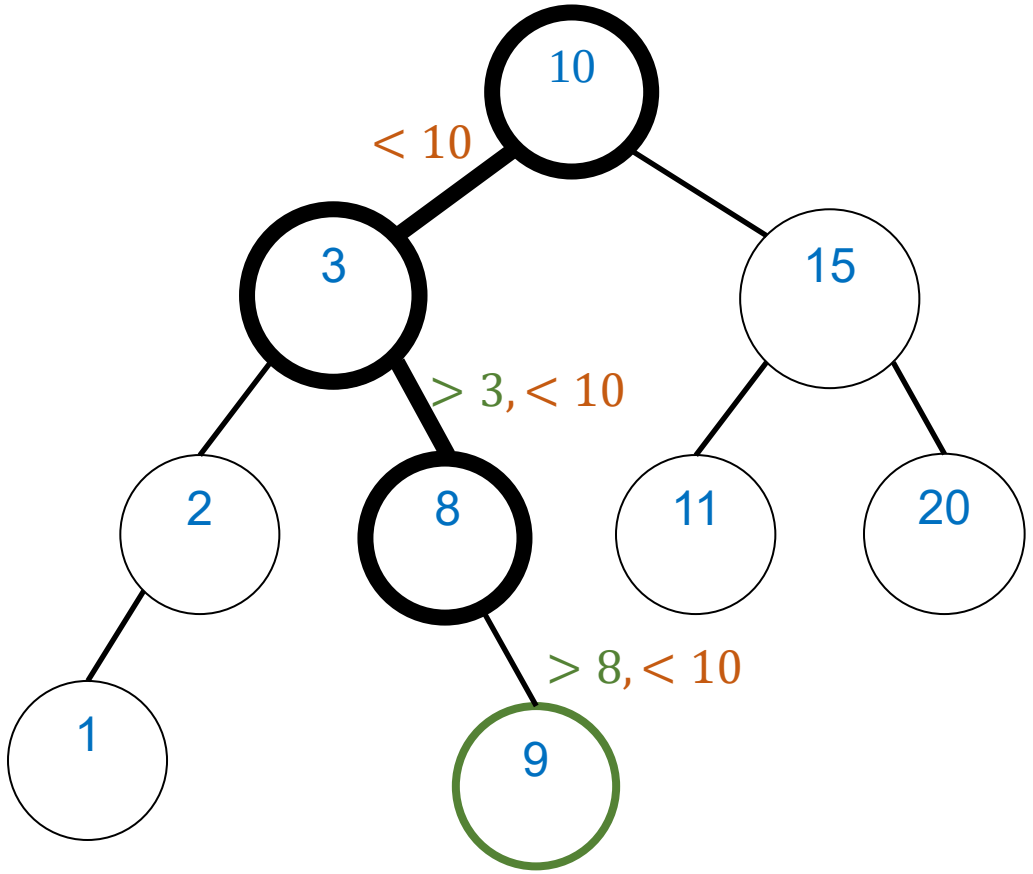


הכנסה לעץ חיפוש בינארי

נכניס לעץ את המפתח 9 – זהו מפתח שאינו קיים בעץ

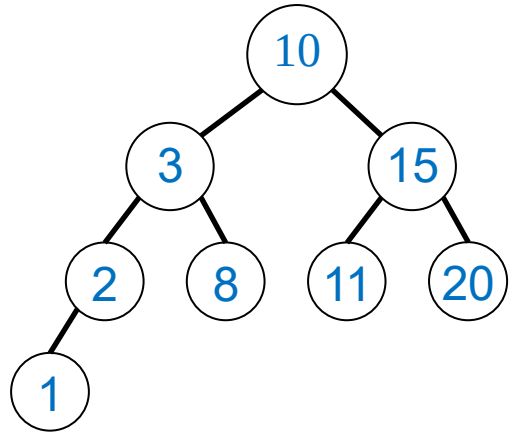
9 גדול מ 8 ולכן נמשיך לתת העץ ששורשו הבן הימני

משום שלצומת זה אין בן ימני, זה יהיה המיקום של הצומת החדש

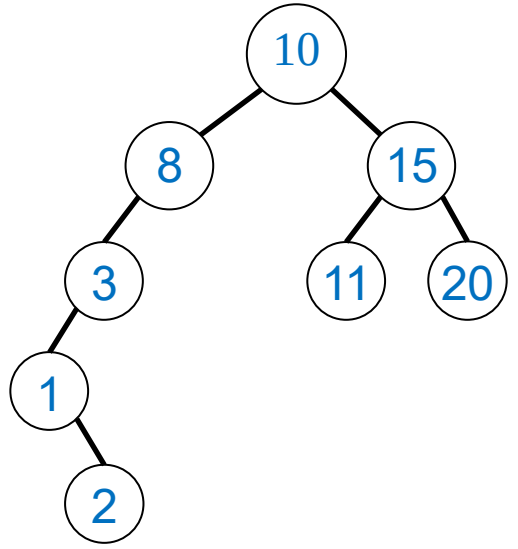


סדר ההכנסה של המפתחות משפיע על מבנה העץ

10, 3, 15, 20, 11, 2, 8, 1



10, 8, 3, 15, 20, 11, 1, 2



Inorder walk

- הילוך בעץ: פרוצדורה שמבקרת בכל צמתי העץ (בכל צומת נדפיס את המפתח בצומת)
- הילוך inorder: לשורש v:
  - הדפס רקורסיבית $v.left$
  - הדפס את $v.key$
  - הדפס רקורסיבית $v.right$

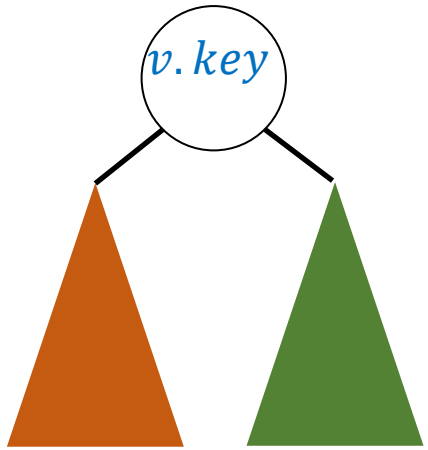
```
def inorder(self):
    '''prints the keys of the tree in a sorted order'''
    def inorder_rec(node):
        if node == None:
            return
        inorder_rec(node.left)
        print(node.key)
        inorder_rec(node.right)

    inorder_rec(self.root)
```

הילוך inorder בעץ ידפיס את מפתחות העץ מקטן לגדול

נוכיח באינדוקציה (שלמה) על גודל העץ (=מספר הצמתים בעץ).

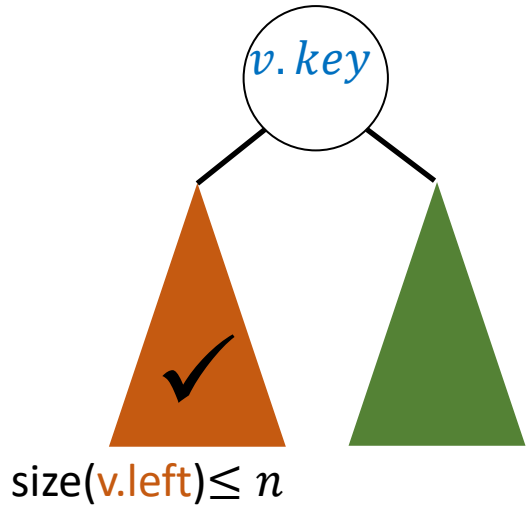
- בסיס: n=1, טריוויאלי
- נניח שמתקיים לכל עץ בעל i צמתים כאשר $i \leq n$
- עבור עץ בעל $n+1$ צמתים:



הילוך inorder בעץ ידפיס את מפתחות העץ מקטן לגדול

נוכיח באינדוקציה (שלמה) על גודל העץ (=מספר הצמתים בעץ).

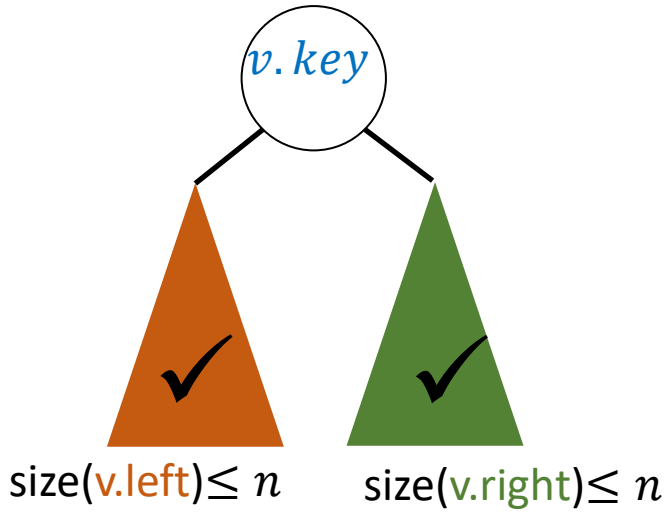
- בסיס: n=1, טריוויאלי
- נניח שמתקיים לכל עץ בעל i צמתים כאשר $i \leq n$
- עבור עץ בעל $n+1$ צמתים:



הילוך inorder בעץ ידפיס את מפתחות העץ מקטן לגדול

נוכיח באינדוקציה (שלמה) על גודל העץ (=מספר הצמתים בעץ).

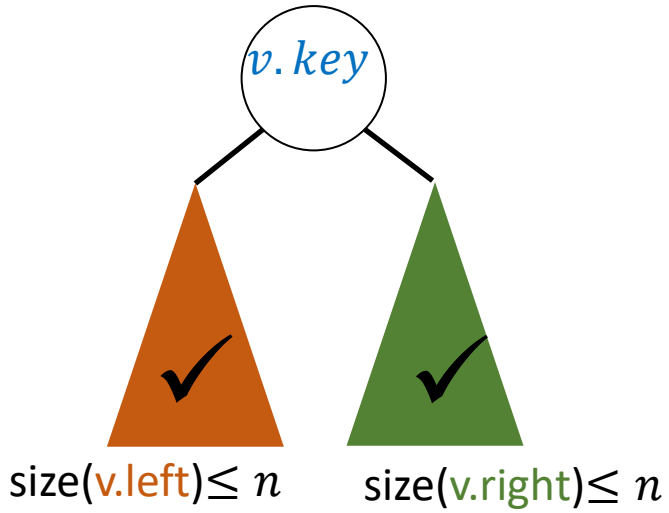
- בסיס: n=1, טריוויאלי
- נניח שמתקיים לכל עץ בעל i צמתים כאשר $i \leq n$
- עבור עץ בעל $n+1$ צמתים:



הילוך inorder בעץ ידפיס את מפתחות העץ מקטן לגדול

נוכיח באינדוקציה (שלמה) על גודל העץ (=מספר הצמתים בעץ).

- בסיס: n=1, טריוויאלי
- נניח שמתקיים לכל עץ בעל i צמתים כאשר $i \leq n$
- עבור עץ בעל $n+1$ צמתים:



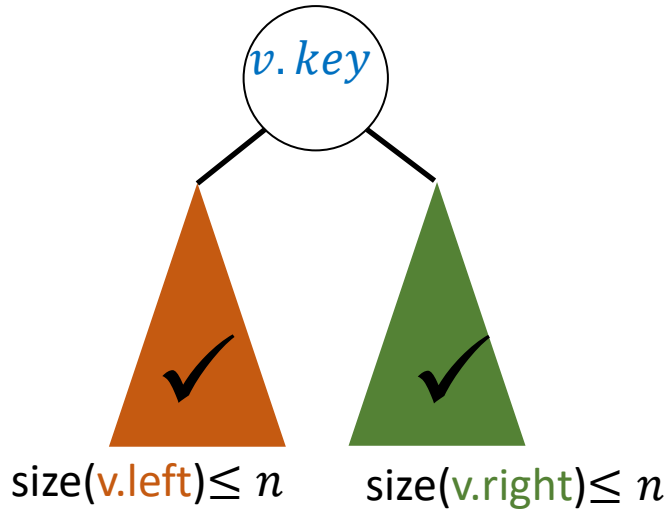
לפי האינווריאנטה של עץ חיפוש בינארי מתקיים:

$$keys(v.left) < v.key < keys(v.right)$$

הילוך inorder בעץ ידפיס את מפתחות העץ מקטן לגדול

נוכיח באינדוקציה (שלמה) על גודל העץ (=מספר הצמתים בעץ).

- בסיס: n=1, טריוויאלי
- נניח שמתקיים לכל עץ בעל i צמתים כאשר $i \leq n$
- עבור עץ בעל $n+1$ צמתים:



לפי האינווריאנטה של עץ חיפוש בינארי מתקיים:

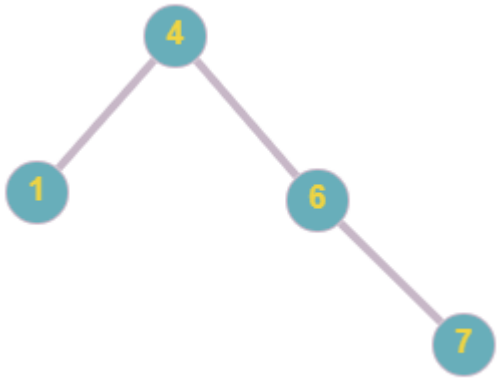
$$keys(v.left) < v.key < keys(v.right)$$

בשילוב עם סדר ההדפסה של inorder נקבל
שהמפתחות בעץ יודפסו מקטן לגדול

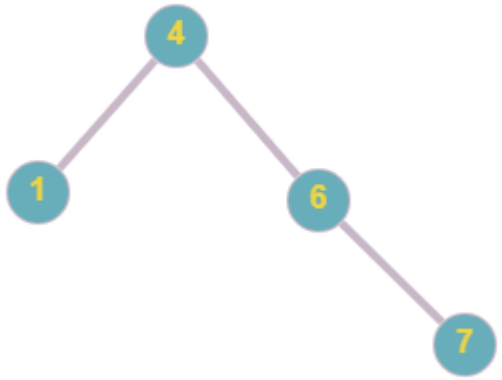
רשימות שמייצרות עצים זהים

נאמר ששתי רשימות מייצרות את אותו עץ אם העצים שנוצרים מהכנסת איברי הרשימות הם בעלי תוכן ומבנה זהה.

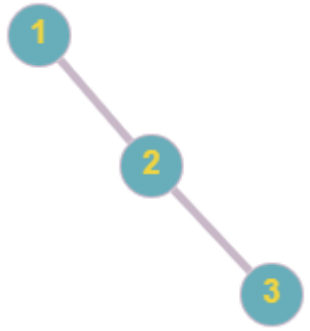
[1,3,2],[1,2,3] לא. אבל [4,1,6,7] ו-[4,6,1,7] כן:



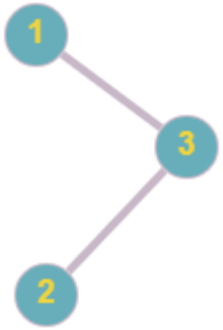
[4, 6, 1, 7]



[4, 1, 6, 7]



[1, 2, 3]



[1, 3, 2]

רשימות שמייצרות עצים זהים

- בהינתן שתי רשימות, נרצה להכריע האם הן מייצרות את אותו העץ.
- תנאי עצירה:
  - שתי רשימות ריקות מייצרות את אותו עץ
  - שתי רשימות באורכים שונים לא מייצרות את אותו עץ
  - שתי רשימות עם איבר ראשון שונה לא מייצרות את אותו עץ (למה?)
- תובנות:
  - שורש = ראש הרשימה
  - תת-עץ שמאלי = תת הרשימה שקטנה מראש הרשימה
  - תת-עץ ימני = תת הרשימה שגדולה מראש הרשימה
- צעד הרקורסיה: נבדוק האם תתי הרשימות שקטנות מהשורש יוצרות אותו עץ **<u>וגם</u>** תתי הרשימה שגדולות מהשורש

רשימות שמייצרות עצים זהים

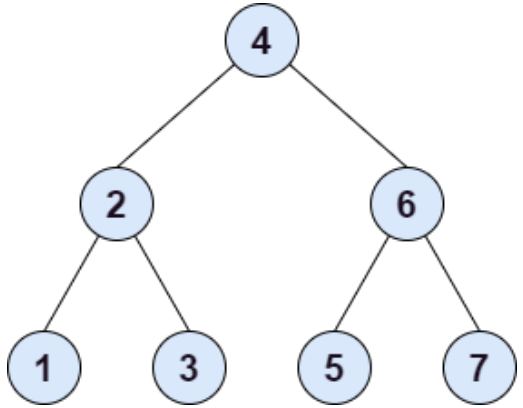
```python
def same_tree(lst1, lst2):
    if len(lst1) == len(lst2) == 0:
        return True
    if len(lst1) != len(lst2) or lst1[0] != lst2[0]:
        return False
    sm1 = [x for x in lst1 if x < lst1[0]]
    lg1 = [x for x in lst1 if x > lst1[0]]
    sm2 = [x for x in lst2 if x < lst2[0]]
    lg2 = [x for x in lst2 if x > lst2[0]]
    small = same_tree(sm1, sm2)
    large = same_tree(lg1, lg2)
    return small and large
```

תנאי עצירה

צעד הרקורסיה

בניית עץ מאוזן

בהינתן n, נרצה לבנות עץ חיפוש בינארי מלא על המפתחות $[1, \ldots, 2^n - 1]$.
נשים לב – לכל n קיים עץ יחיד כזה. למשל, עבור $n = 3$:



הרעיון: נתחזק שני אינדקסים שמייצגים את "קצות" העץ הנוכחי (מוכר?)

בניית עץ מאוזן

- הרעיון: נתחזק שני אינדקסים שמייצגים את "קצות" העץ הנוכחי
- בהתחלה: $first = 1, last = 2^n - 1$
- תנאי עצירה: אם $first = last$, צומת בודד עם מפתח $first$
- אחרת, נחשב $mid = \frac{first+last}{2}$
- נבנה צומת mid ונוסיף לה בנים:
  - משמאל, $first, mid - 1$
  - מימין, $mid + 1, last$

בניית עץ מאוזן

```
3     def insert_balanced(self, n):
4         def insert_balanced_rec(first, last):
5             if first == last:
6                 return Tree_node(first, first)
7             mid = (first + last) // 2
8             root = Tree_node(mid, mid)
9             root.left = insert_balanced_rec(first, mid - 1)
10            root.right = insert_balanced_rec(mid + 1, last)
11            return root
12        self.root = insert_balanced_rec(1, 2**n - 1)
```

תנאי עצירה

צעד הרקורסיה

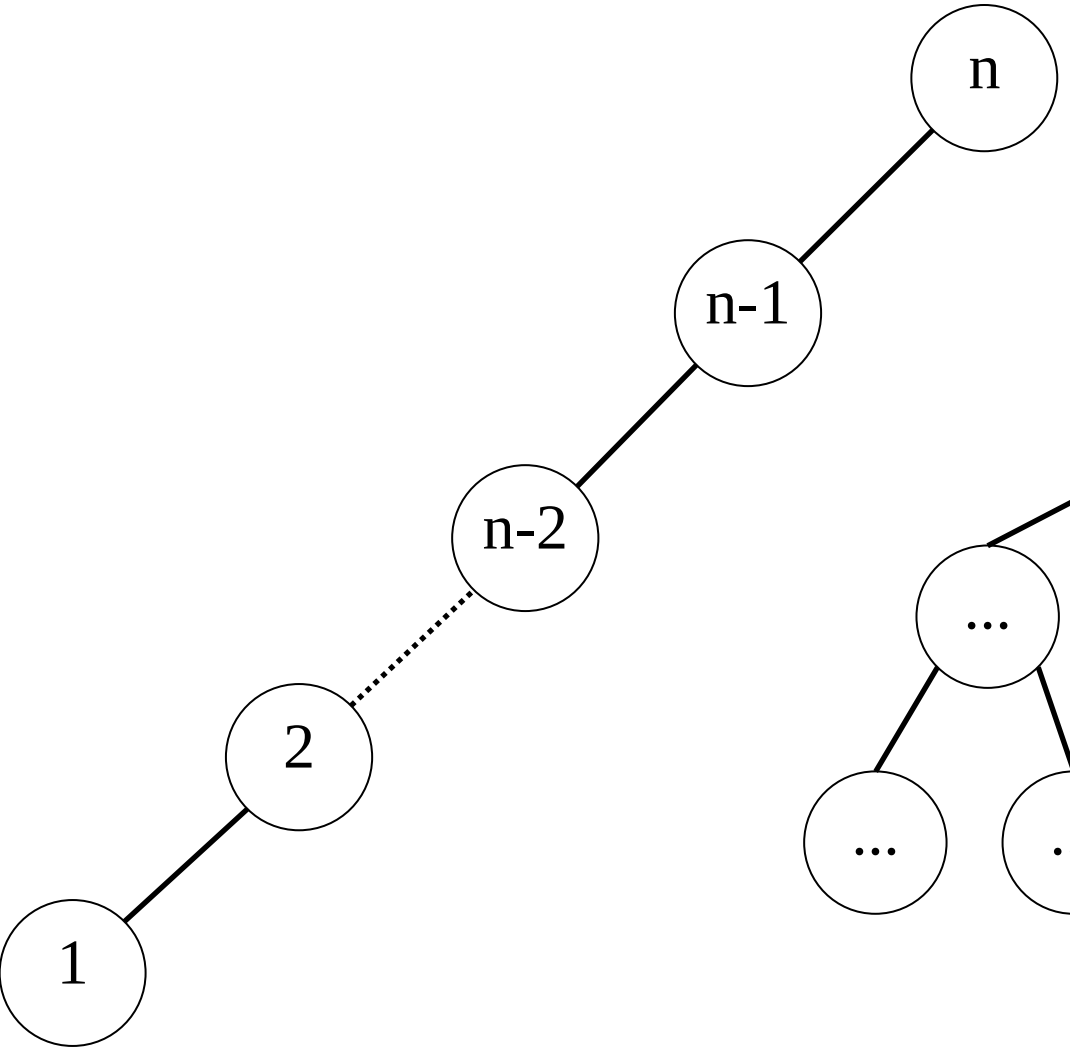
מעטפת

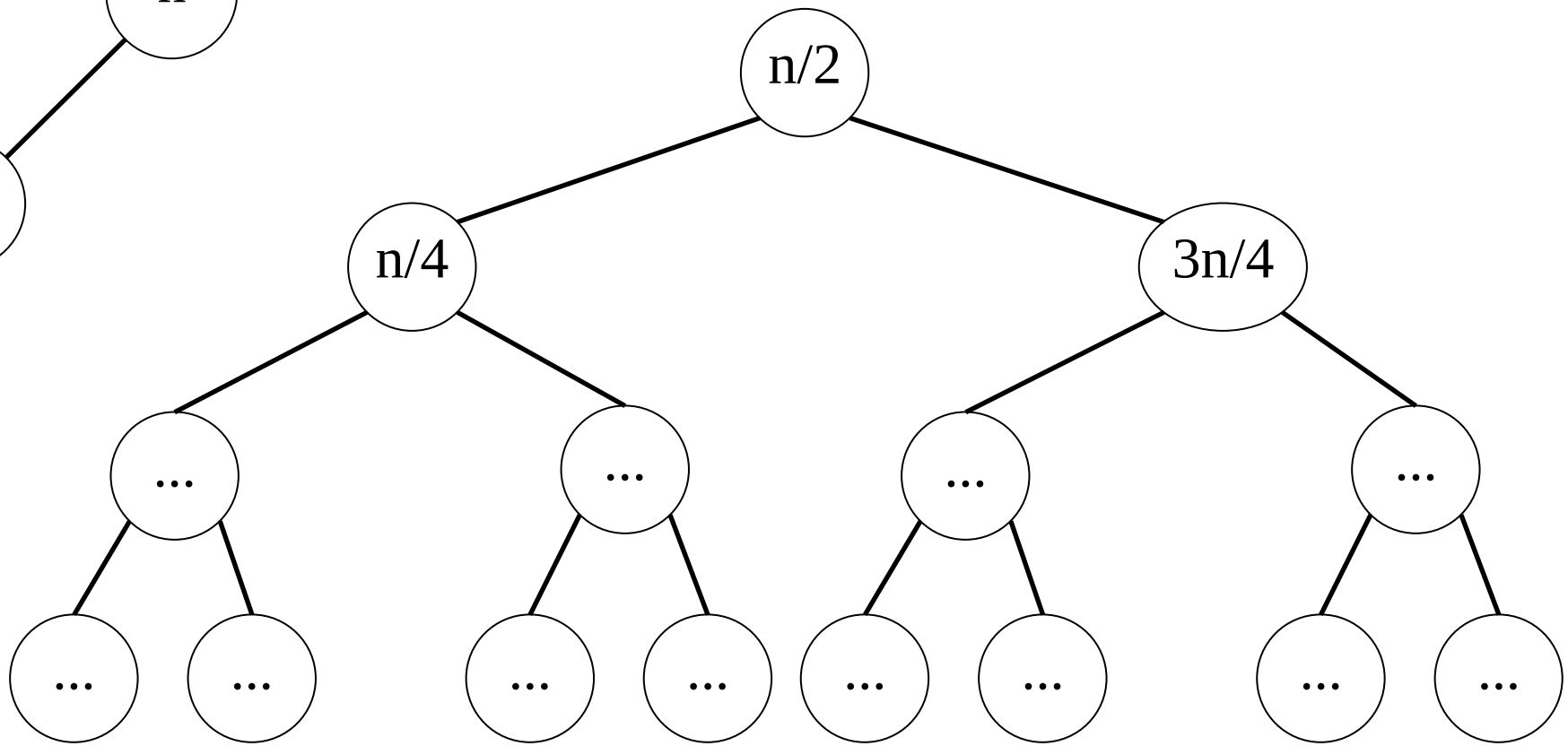
עומק עץ חיפוש בינארי בעל n איברים ולמה זה מעניין

WC

BC

Avg.?





- נרצה לממש מאגר ת"ז של סטודנטים בת"א
- מה צריך: הכנסה וחיפוש

- נסמן ב U את קבוצת כל המחרוזות באורך 9 של ספרות, $|U| = 10^9$
- נסמן ב S את קבוצת כל המחרוזות באורך 9 שמהוות תעודות זהות של סטודנטים בת"א.
- נסמן $n = |S|$
- מתקיים:
  - $S \subset U$
  - $|S| \ll |U|$

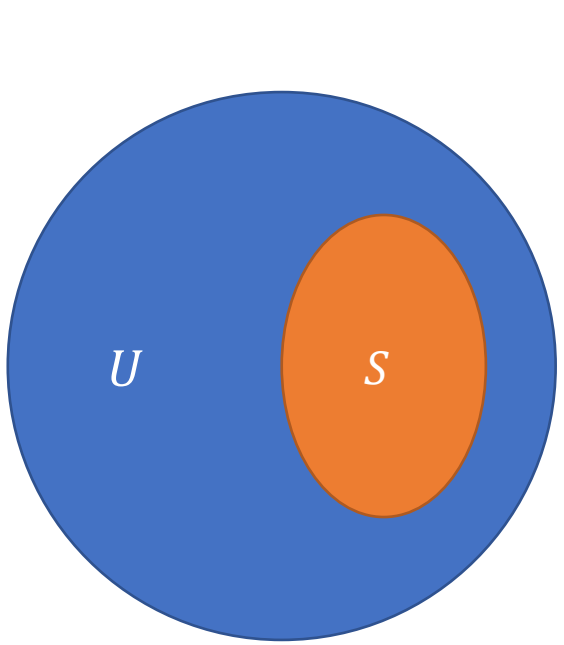
- נרצה לממש מאגר ת"ז של סטודנטים בת"א
- מה צריך: הכנסה וחיפוש

- נאתחל רשימת אפסים L באורך $|U| = 10^9$
- הכנסה: L[x]=1
- חיפוש: L[x]==1?
- זמן:
  - אתחול: $O(|U|)$
  - חיפוש, הכנסה: $O(1)$
- זכרון: $O(|U|)$
- אופטימלי בחיפוש והכנסה. בזבזני בזיכרון ואתחול יקר **<u>מאוד</u>**

- נאתחל BST
- הכנסה וחיפוש כרגיל
- זמן:
  - אתחול: $O(1)$
  - חיפוש, הכנסה: $O(\log n)$
- זכרון: $O(n)$
- אופטימלי באתחול וזיכרון, חיפוש והכנסה "סבירים"

Can we have both?

Hash tables



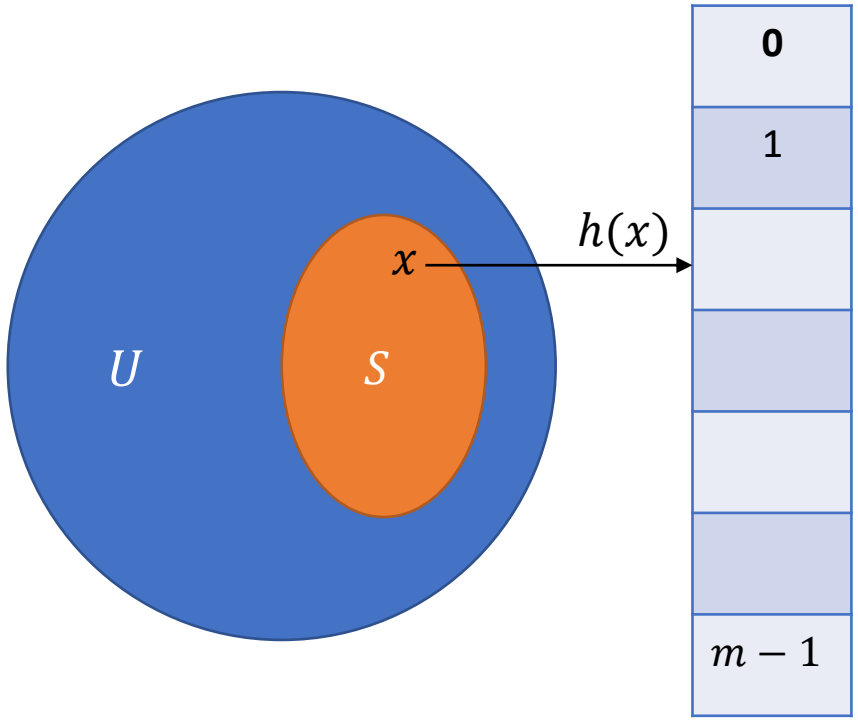
U = כל המחרוזות באורך 9 של ספרות $= \{\{0,1,\dots,9\}^9\}$

$S \subset U$: תעודות הזהות של סטודנטים באוניברסיטה

$n = |S| \ll |U|$

- ניצור טבלה בגודל m
- נמפה איברים מתוך U לתאי הטבלה
  - באמצעות פונקצייה $h: U \to \{0,\dots,m-1\}$
    - למשל: $m = 10, h(x) = x \,\%\, 10$
- אם $x \neq y$ אבל $h(x) = h(y)$ נאמר שהמפתחות מתנגשים
- טענה: אם $|S| > m$ יש התנגשויות
- מה נעשה במקרה כזה?
  - בכל תא רשימת האיברים שמופו אליו

Hash tables

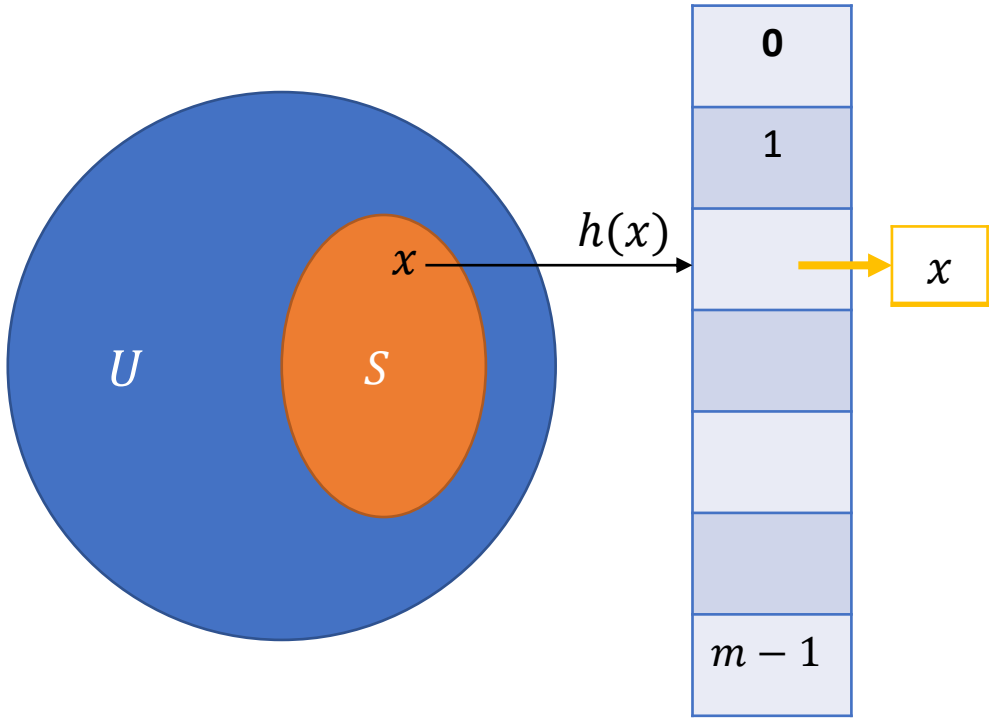


U = כל המחרוזות באורך 9 של ספרות $= \{\{0,1,\dots,9\}^9\}$

$S \subset U$: תעודות הזהות של סטודנטים באוניברסיטה

$$n = |S| \ll |U|$$

Hash tables

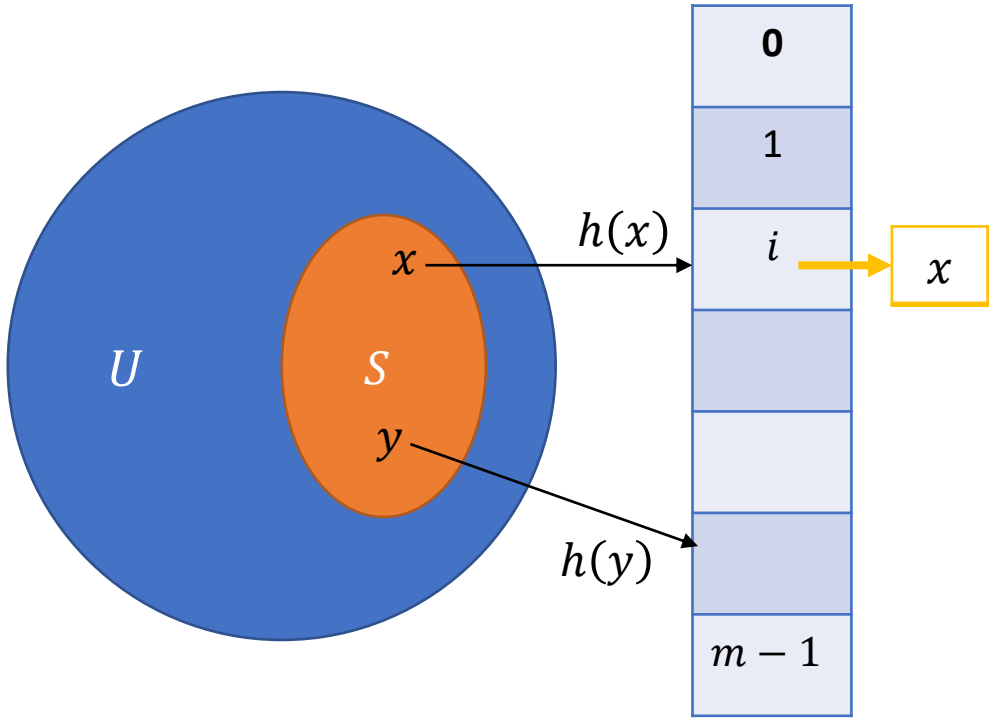


U = כל המחרוזות באורך 9 של ספרות $= \{\{0,1,\dots,9\}^9\}$

$S \subset U$: תעודות הזהות של סטודנטים באוניברסיטה

$$n = |S| \ll |U|$$

Hash tables

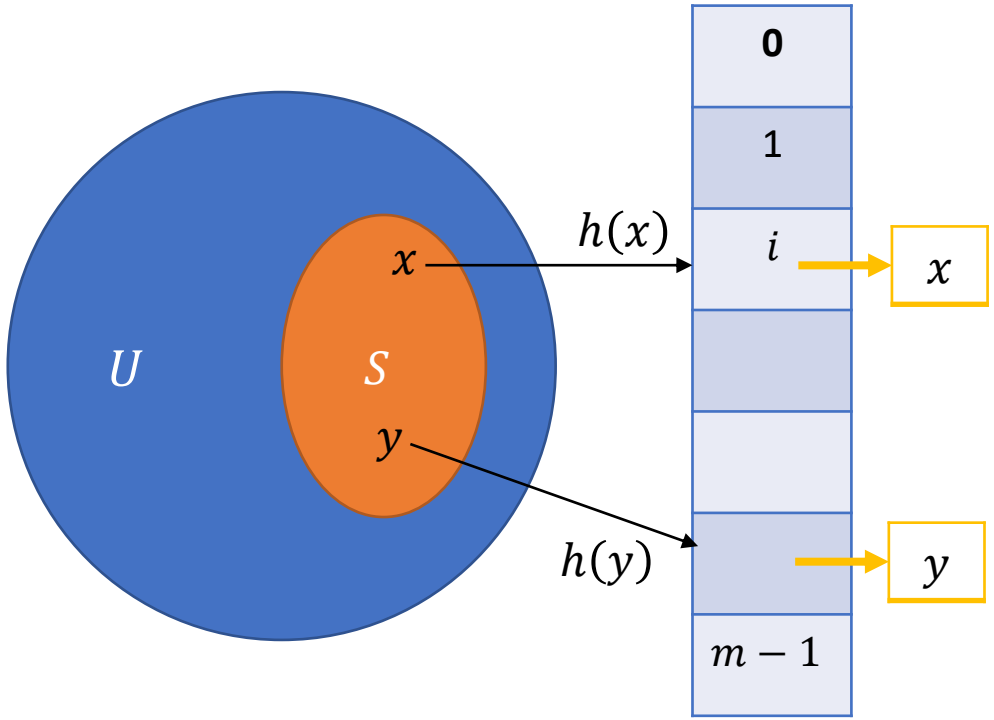


U = כל המחרוזות באורך 9 של ספרות $= \{\{0,1,\dots,9\}^9\}$

$S \subset U$: תעודות הזהות של סטודנטים באוניברסיטה

$$n = |S| \ll |U|$$

Hash tables

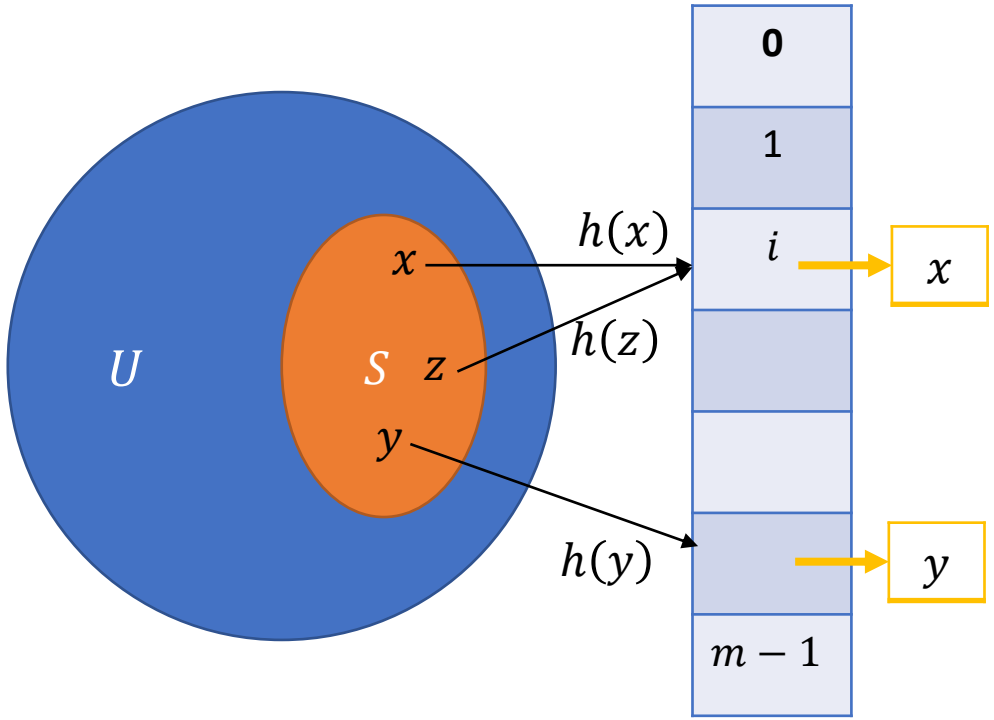


U = כל המחרוזות באורך 9 של ספרות $= \{\{0,1,\dots,9\}^9\}$

$S \subset U$: תעודות הזהות של סטודנטים באוניברסיטה

$$n = |S| \ll |U|$$

Hash tables

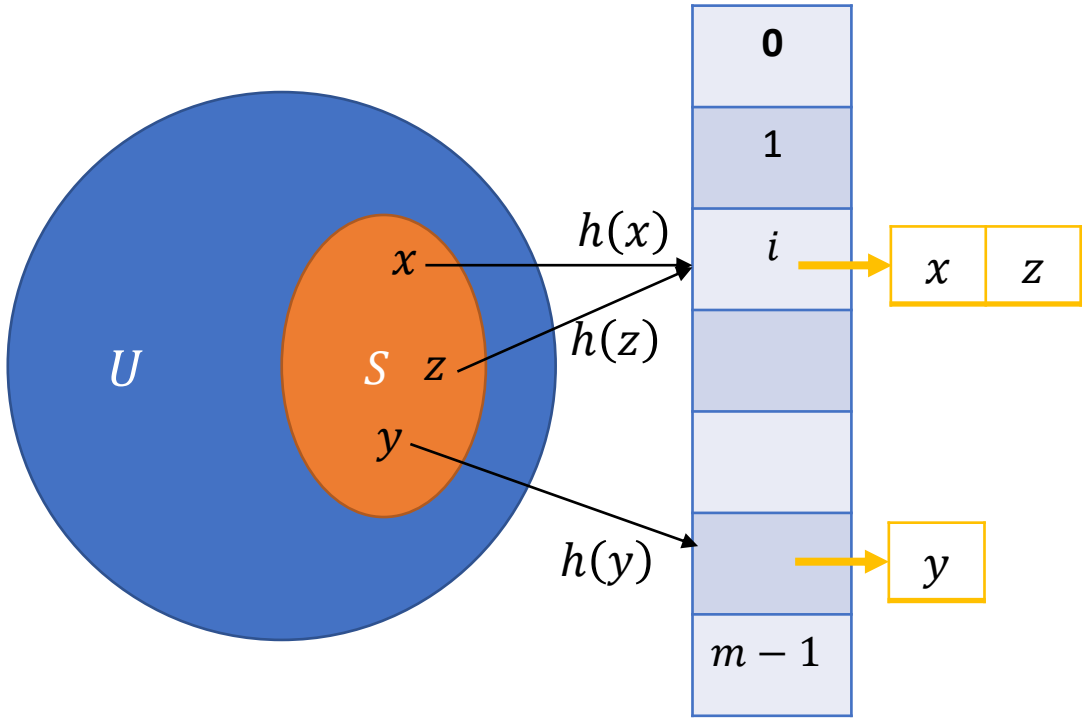


U = כל המחרוזות באורך 9 של ספרות $= \{\{0,1,\ldots,9\}^9\}$

$S \subset U$: תעודות הזהות של סטודנטים באוניברסיטה

$$n = |S| \ll |U|$$

Hash tables

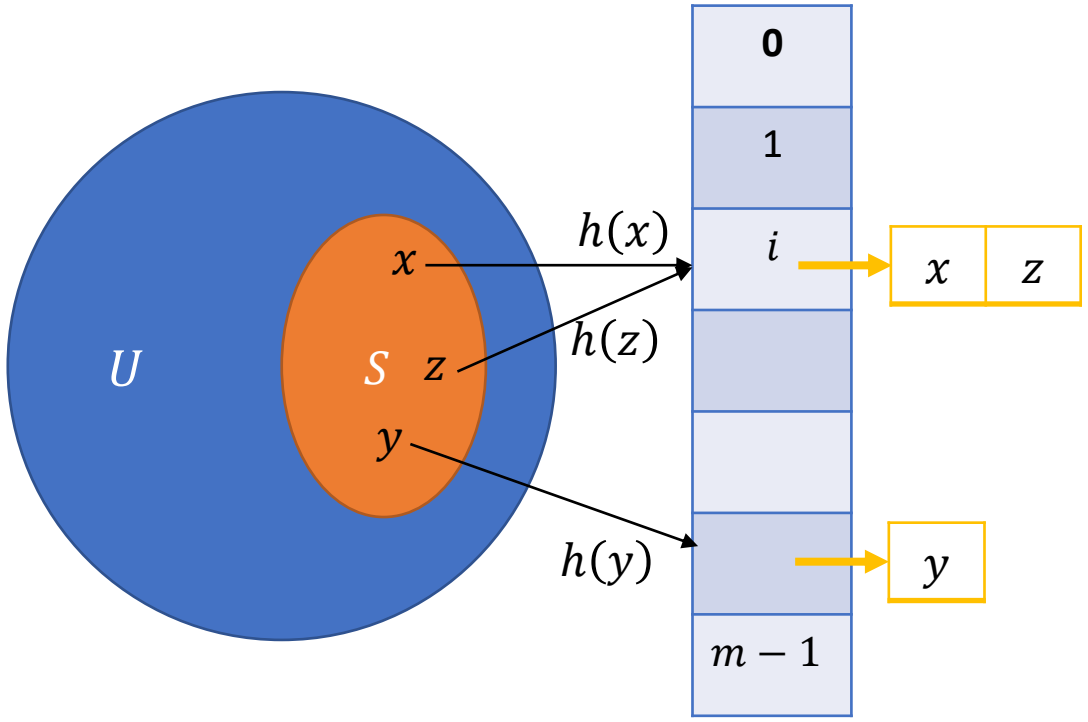


U = כל המחרוזות באורך 9 של ספרות $= \{\{0,1,\ldots,9\}^9\}$

$S \subset U$: תעודות הזהות של סטודנטים באוניברסיטה

$$n = |S| \ll |U|$$

The function h



U = כל המחרוזות באורך 9 של ספרות $= \{\{0,1,\dots,9\}^9\}$

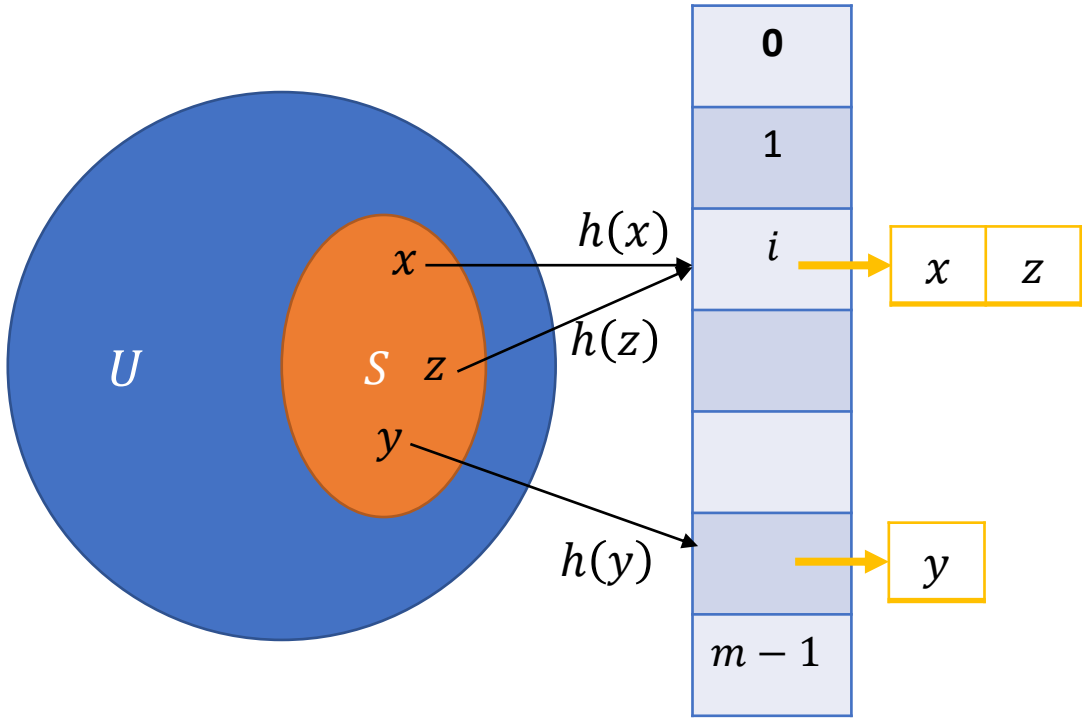
$S \subset U$: תעודות הזהות של סטודנטים באוניברסיטה

$$n = |S| \ll |U|$$

דרישות מ h:

- קלה לחישוב (סיבוכיות זמן לינארית בגודל הקלט שלה)
- דטרמיניסטית
- תפזר את איברי U בצורה "טובה" על פני תאי הטבלה
- מה זה פיזור טוב?

Hash tables: find



U = כל המחרוזות באורך 9 של ספרות $= \{\{0,1,\dots,9\}^9\}$

$S \subset U$: תעודות הזהות של סטודנטים באוניברסיטה

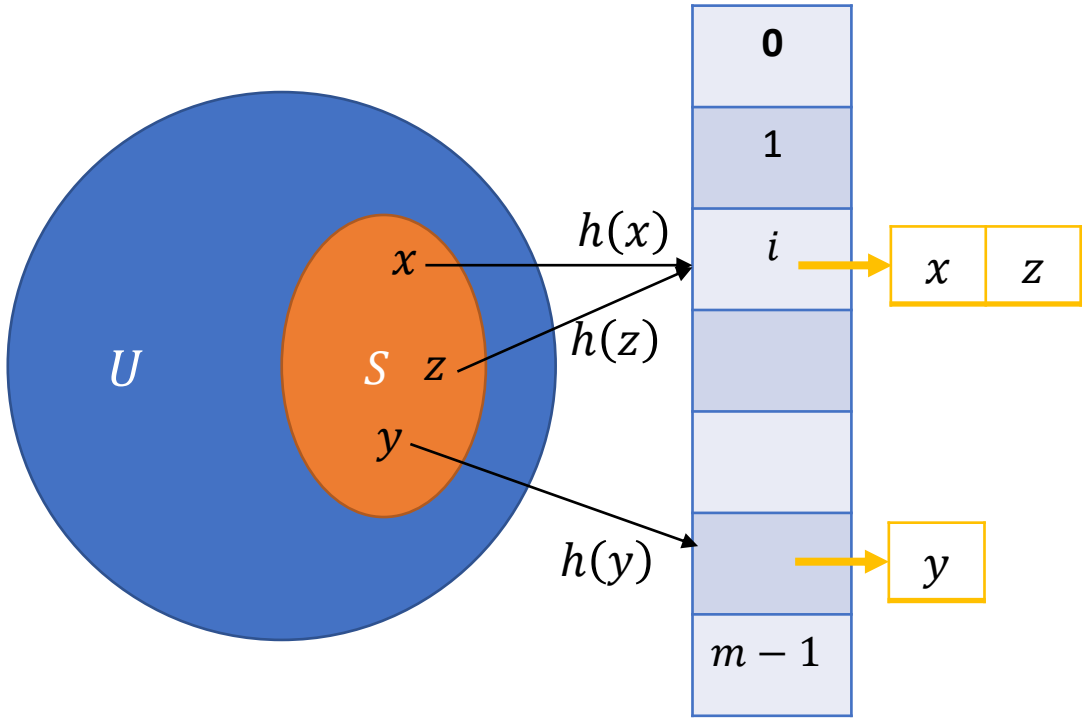
$$n = |S| \ll |U|$$

נחפש את המפתח q בטבלה:

שלב 1: נחשב את $h(q)$. נניח שהתוצאה היא i

שלב 2: נעבור על רשימת האיברים בתא באינדקס i, ונשווה איבר איבר ל q. אם מצאנו איבר שהמפתח שלו זהה ל q נעצור ונחזיר את האיבר.

סיבוכיות ממוצעת של חיפוש



- m גודל הטבלה
- n מספר האיברים במבנה הנתונים
- k גודל מפתח (לעיתים $k = O(1)$)
- $\alpha = \frac{n}{m}$ פקטור העומס = המספר הממוצע של איברים בתא

- ננית ש h פועלת בסיבוכיות לינארית בגודל הקלט שלה (המפתח)

סיבוכיות ממוצעת של חיפוש

- m גודל הטבלה
- n מספר האיברים שנשמור
- k גודל מפתח (לעיתים $k = O(1)$)
- $\alpha = \frac{n}{m}$ פקטור העומס = המספר הממוצע של איברים בתא
- נניח ש h פועלת בסיבוכיות לינארית בגודל הקלט שלה

חיפוש q:
שלב 1: נחשב את $h(q)$. נניח שהתוצאה היא i
שלב 2: נעבור על רשימת האיברים בתא באינדקס i, ונשווה איבר איבר ל q. אם מצאנו איבר שהמפתח שלו זהה ל q נעצור ונחזיר את האיבר.

סיבוכיות שלב 1: $O(k)$ הפעלת h על מפתח בגודל k

סיבוכיות שלב 2: $O(\alpha k)$

- בממוצע יהיו α איברים בתא. עלות השוואה בין זוג מפתחות $O(k)$

סה"כ סיבוכיות חיפוש ממוצעת: $O\big((1+\alpha)k\big)$

- אם $k = O(1)$ אז הסיבוכיות הממוצעת היא $O(1+\alpha)$
- אם בנוסף $m = cn$ כאשר $c > 0$ הינו קבוע, אז הסיבוכיות הממוצעת היא: $O(1)$

סיבוכיות worst-case של חיפוש

- m גודל הטבלה
- n מספר האיברים שנשמור
- k גודל מפתח (לעיתים $k = O(1)$)
- $\alpha = \frac{n}{m}$ פקטור העומס = המספר הממוצע של איברים בתא
- נניח ש h פועלת בסיבוכיות לינארית בגודל הקלט שלה

חיפוש q:
שלב 1: נחשב את $h(q)$. נניח שהתוצאה היא i
שלב 2: נעבור על רשימת האיברים בתא באינדקס i, ונשווה איבר איבר ל q. אם מצאנו איבר שהמפתח שלו זהה ל q נעצור ונחזיר את האיבר.

סיבוכיות שלב 1: $O(k)$ הפעלת h על מפתח בגודל k

סיבוכיות שלב 2: $O(nk)$

- במקרה הגרוע כל n האיברים יהיו באותו התא.

סה"כ סיבוכיות חיפוש: $O(nk)$

- אם $k = O(1)$ אז הסיבוכיות היא $O(n)$

Repeating substring

- הקלט:
  - מחרוזת st (באורך n)
  - מספר שלם $0 \leq \ell \leq n$
- האם ב st ישנה תת מחרוזת (רצופה) באורך ℓ שמופיעה יותר מפעם אחת?

- דוגמאות:

st = “abcabcabc” , $\ell = 3$ → True

st = “abcabcabc” , $\ell = 6$ → True

st = “abcabcabc” , $\ell = 7$ → False

פתרון נאיבי

- נשווה כל תת מחרוזת באורך ℓ לתתי המחרוזות באורך ℓ שמופיעות אחריה במחרוזת st (מתחילות באינדקס גדול יותר)

st =

a	b	c	d	b	c	d	f

$\ell = 3$

פתרון נאיבי

- נשווה כל תת מחרוזת באורך ℓ לתתי המחרוזות באורך ℓ שמופיעות אחריה במחרוזת st (מתחילות באינדקס גדול יותר)

st =

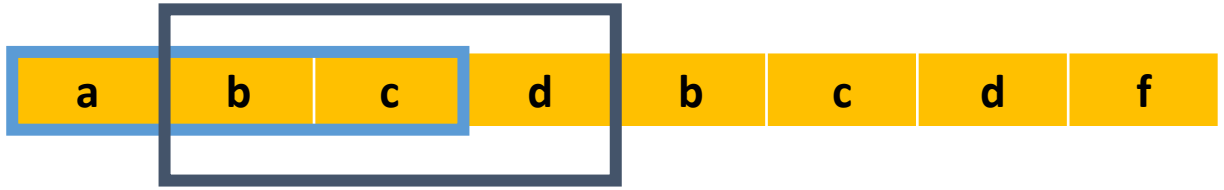
a	b	c	d	b	c	d	f

$\ell = 3$

פתרון נאיבי

- נשווה כל תת מחרוזת באורך ℓ לתתי המחרוזות באורך ℓ שמופיעות אחריה במחרוזת st (מתחילות באינדקס גדול יותר)

st =

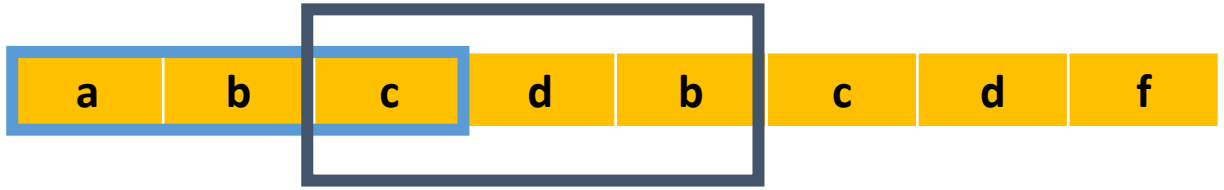


$\ell = 3$

פתרון נאיבי

- נשווה כל תת מחרוזת באורך ℓ לתתי המחרוזות באורך ℓ שמופיעות אחריה במחרוזת st (מתחילות באינדקס גדול יותר)

st =

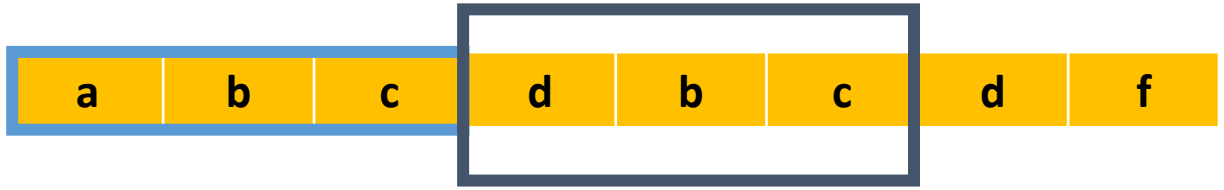


$\ell = 3$

פתרון נאיבי

- נשווה כל תת מחרוזת באורך ℓ לתתי המחרוזות באורך ℓ שמופיעות אחריה במחרוזת st (מתחילות באינדקס גדול יותר)

st =

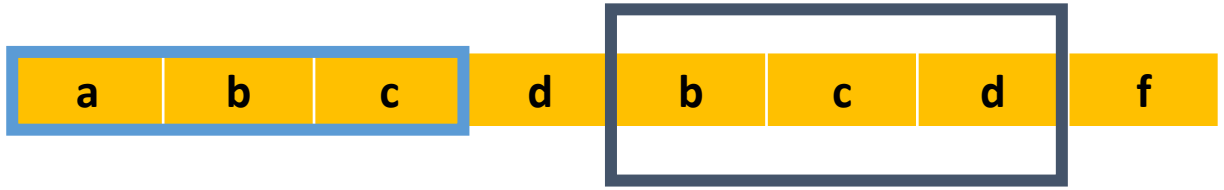


$\ell = 3$

פתרון נאיבי

- נשווה כל תת מחרוזת באורך ℓ לתתי המחרוזות באורך ℓ שמופיעות אחריה במחרוזת st (מתחילות באינדקס גדול יותר)

st =

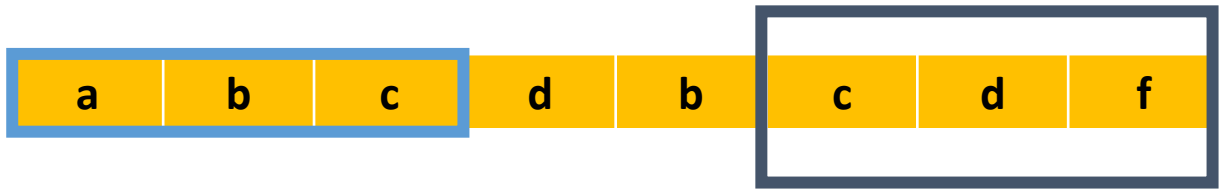


$\ell = 3$

פתרון נאיבי

- נשווה כל תת מחרוזת באורך ℓ לתתי המחרוזות באורך ℓ שמופיעות אחריה במחרוזת st (מתחילות באינדקס גדול יותר)

st =



$\ell = 3$

פתרון נאיבי

- נשווה כל תת מחרוזת באורך ℓ לתתי המחרוזות באורך ℓ שמופיעות אחריה במחרוזת st (מתחילות באינדקס גדול יותר)

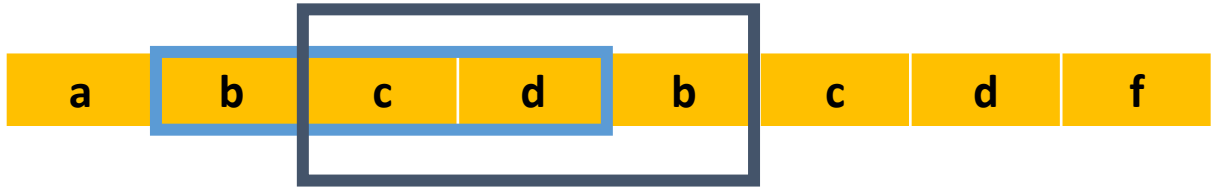
st =

$\ell = 3$

פתרון נאיבי

- נשווה כל תת מחרוזת באורך ℓ לתתי המחרוזות באורך ℓ שמופיעות אחריה במחרוזת st (מתחילות באינדקס גדול יותר)

st =

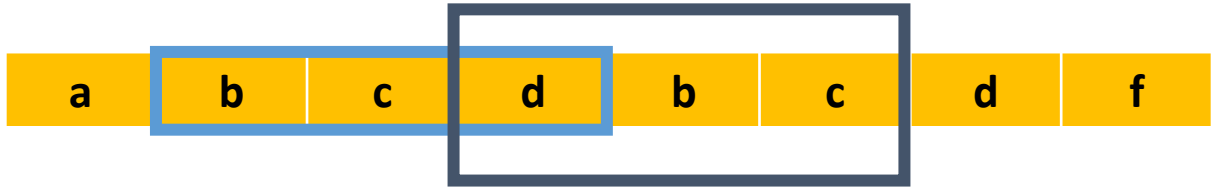


$\ell = 3$

פתרון נאיבי

- נשווה כל תת מחרוזת באורך ℓ לתתי המחרוזות באורך ℓ שמופיעות אחריה במחרוזת st (מתחילות באינדקס גדול יותר)

st =



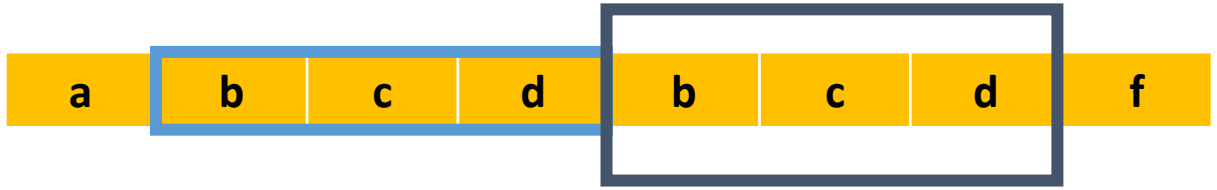
$\ell = 3$

פתרון נאיבי

- נשווה כל תת מחרוזת באורך ℓ לתתי המחרוזות באורך ℓ שמופיעות אחריה במחרוזת st (מתחילות באינדקס גדול יותר)

st =

$\ell = 3$



✓ It’s a match!

פתרון שמשתמש בטבלת Hash

- נעבור על כל תתי המחרוזות באורך ℓ. נתחזק טבלת Hash. לכל תת מחרוזת – נבדוק האם נמצאת בטבלה. אם כן נחזיר True. אם לא נכניס לטבלה.

st =

$\ell = 3$

תוכן טבלת ה Hash:

פתרון שמשתמש בטבלת Hash

- נעבור על כל תתי המחרוזות באורך ℓ. נתחזק טבלת Hash. לכל תת מחרוזת – נבדוק האם נמצאת בטבלה. אם כן נחזיר True. אם לא נכניס לטבלה.

st =

$\ell = 3$

תוכן טבלת ה Hash:

abc

פתרון שמשתמש בטבלת Hash

- נעבור על כל תתי המחרוזות באורך ℓ. נתחזק טבלת Hash. לכל תת מחרוזת – נבדוק האם נמצאת בטבלה. אם כן נחזיר True. אם לא נכניס לטבלה.

st =

$\ell = 3$

תוכן טבלת ה Hash:

abc

פתרון שמשתמש בטבלת Hash

- נעבור על כל תתי המחרוזות באורך ℓ. נתחזק טבלת Hash. לכל תת מחרוזת – נבדוק האם נמצאת בטבלה. אם כן נחזיר True. אם לא נכניס לטבלה.

st =

$\ell = 3$

תוכן טבלת ה Hash:

abc, bcd

פתרון שמשתמש בטבלת Hash

- נעבור על כל תתי המחרוזות באורך ℓ. נתחזק טבלת Hash. לכל תת מחרוזת – נבדוק האם נמצאת בטבלה. אם כן נחזיר True. אם לא נכניס לטבלה.

st =

$\ell = 3$

תוכן טבלת ה Hash:

abc, bcd

פתרון שמשתמש בטבלת Hash

- נעבור על כל תתי המחרוזות באורך ℓ. נתחזק טבלת Hash. לכל תת מחרוזת – נבדוק האם נמצאת בטבלה. אם כן נחזיר True. אם לא נכניס לטבלה.

st =

$\ell = 3$

תוכן טבלת ה Hash:

abc, bcd, cdb

פתרון שמשתמש בטבלת Hash

- נעבור על כל תתי המחרוזות באורך ℓ. נתחזק טבלת Hash. לכל תת מחרוזת – נבדוק האם נמצאת בטבלה. אם כן נחזיר True. אם לא נכניס לטבלה.

st =

$\ell = 3$

תוכן טבלת ה Hash:

abc, bcd, cdb

פתרון שמשתמש בטבלת Hash

- נעבור על כל תתי המחרוזות באורך ℓ. נתחזק טבלת Hash. לכל תת מחרוזת – נבדוק האם נמצאת בטבלה. אם כן נחזיר True. אם לא נכניס לטבלה.

st =

$\ell = 3$

תוכן טבלת ה Hash:

abc, bcd, cdb, dbc

פתרון שמשתמש בטבלת Hash

- נעבור על כל תתי המחרוזות באורך ℓ. נתחזק טבלת Hash. לכל תת מחרוזת – נבדוק האם נמצאת בטבלה. אם כן נחזיר True. אם לא נכניס לטבלה.

st =

$\ell = 3$

תוכן טבלת ה Hash:

abc, bcd, cdb, dbc

פתרון שמשתמש בטבלת Hash

- נעבור על כל תתי המחרוזות באורך ℓ. נתחזק טבלת Hash. לכל תת מחרוזת – נבדוק האם נמצאת בטבלה. אם כן נחזיר True. אם לא נכניס לטבלה.

st =

$\ell = 3$

תוכן טבלת ה Hash:

abc, bcd, cdb, dbc

✓ It’s a match!