הנחת יסוד

- ניניח הערכים בעלים

- לצורכי השיעור לא נראה את ערכי המפתחות, חוקיות העץ נשארת אותו דבר (עץ חיפוש בינארי)

2

עצי אדום-שחור: הגדרה

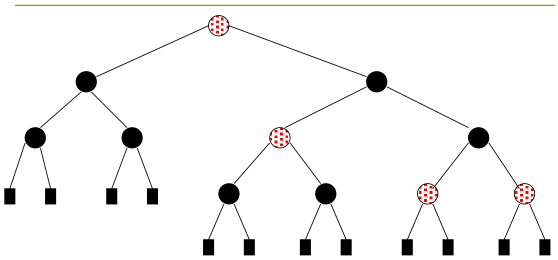
עץ חיפוש בינארי

כל צומת צבועה באדום או **שחור** כך ש:

1. העלים **שחורים**
2. כל מסלול מהשורש לעלה מכיל את אותו מספר של צמתים **שחורים** (החוק ה**שחור**)
3. לכל צומת אדום,אם יש לו אב, האב שחור (החוק האדום)

5

דוגמה

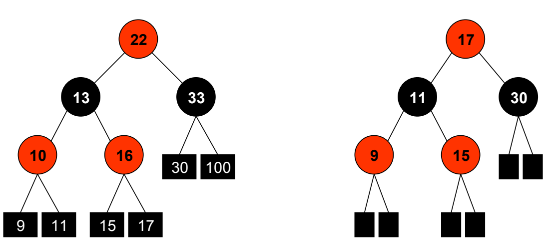


6

תרגיל

- **נוכיח כי עומקו של עץ אדום שחור עם n מפתחות הוא O(logn)**

דוגמאות



עומק שחור = 2

פתרון התרגיל

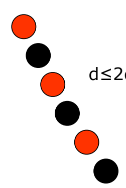
- נניח כי כל הקדקודים שחורים
  - עץ זה חייב להיות **מלא ושלם**
- נניח שהעומק הוא d, כמה קדקודים יש?
  - 2^d-1
- אזי העומק בעץ עם n צמתים הוא $\log(n+1)$

פתרון התרגיל - אינטואיציה

1. מס' קדקודים שחורים = $\Theta(n)$
2. עומק שחור = $\log(\#blacks + 1)$
3. עומק שחור $=\Theta(depth)$
   - ליתר דיוק: עומק שחור $* 2 \le$ העומק

פתרון תרגיל

- <u>למה 1</u>: נסמן ב-d_b את העומק השחור וב-d את העומק, אזי $d \le 2d_b$
  - נתבונן במסלול הכי ארוך, מס' הקדקודים בו d
  - מה מספר השחורים בו?
    - d_b מספר השחורים $\le {}^d/_2$ (המקרה הכי גרוע) ולכן $d \le 2d_b$



- נראה כי $d_b=O(\log n)$:
  - נוכיח כי $d_b=O(\log(n_b))$ כאשר n_b מס' הקדקודים השחורים

פתרון תרגיל

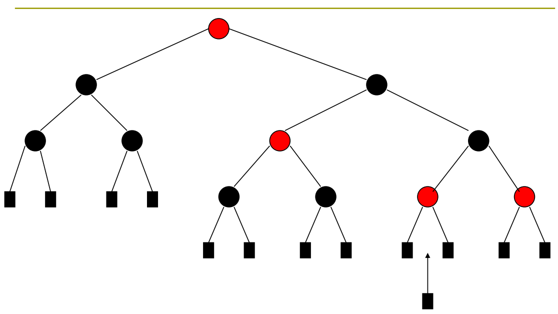
- <u>למה 2:</u> בעץ אדום-שחור שגובהו השחור d_b יש $\le 2^{d_b}-1$ קדקודים שחורים
  - הוכחה באינדוקציה:
  - אם $d_b=1$ יש לפחות קדקוד שחור אחד, $2^1-1=1<d_b$.
  - צעד אינדוקציה: ניקח עץ עם עומק שחור d_b
    - אם השורש שחור:
      - בתת עץ שמאלי עומק שחור d_b-1 ולפי האינדוקציה יש בו $\le 2^{d_b-1}-1$ שחורים
      - גם בימני $\le 2^{d_b-1}-1$ שחורים
      - ביחד עם השורש, בכל העץ יש $\le 2^{d_b}-1 = 2(2^{d_b-1}-1)+1$.
    - אם השורש אדום: אז יש לו בן שחור v, ניקח את תת העץ שהשורש שלו הוא v, אותו טיעון יראה שיש בו $\le 2^{d_b}-1$ שחורים

פתרון תרגיל

- למה 2 הראתה כי: $n_b \ge 2^{d_b}-1$
- ולכן:

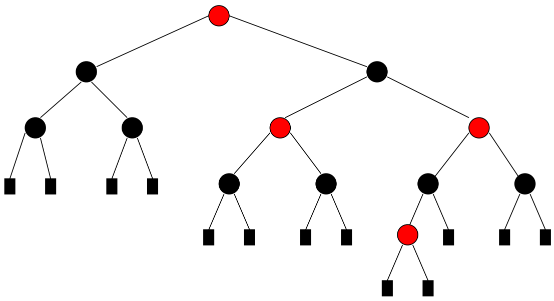
$$\underbrace{{}^d/_2 \le d_b}_{\text{למה 1}} \le \underbrace{\log(n_b+1)}_{\text{למה 2}} \le \underbrace{\log(n+1)}_{\text{טריוויאלי}}$$

Insert

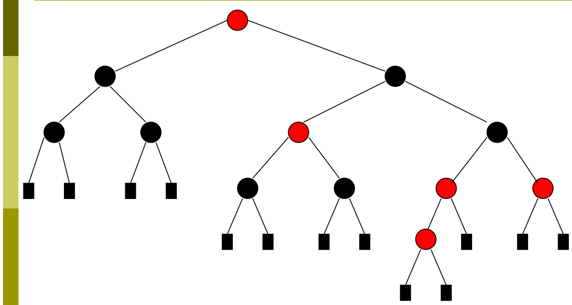


14

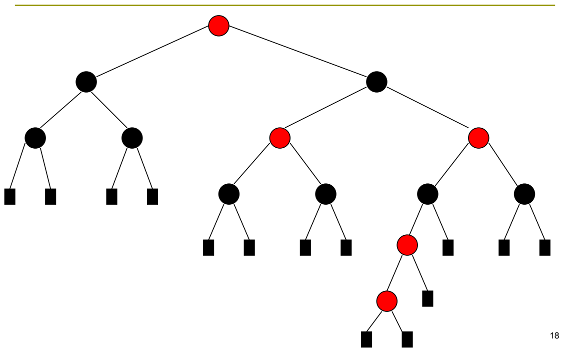
Insert (cont)



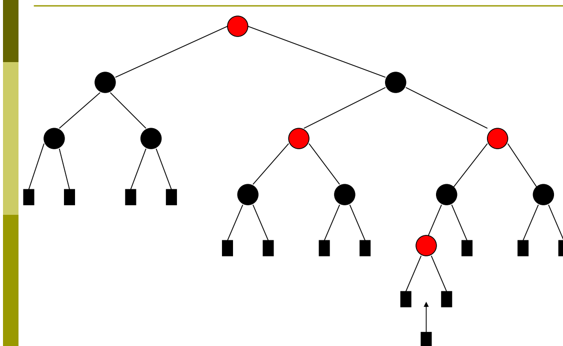
Insert (cont)



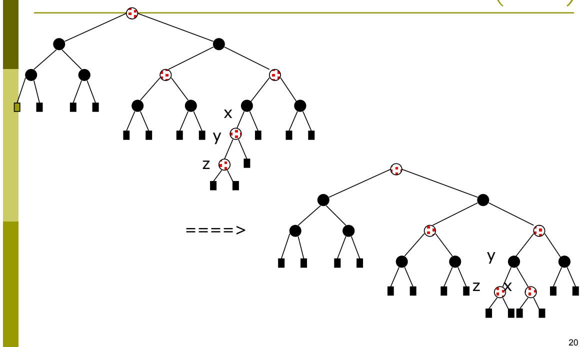
Insert (cont)



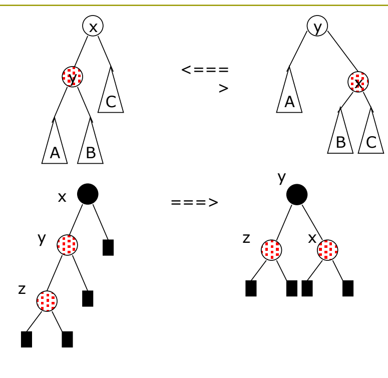
Insert (cont)



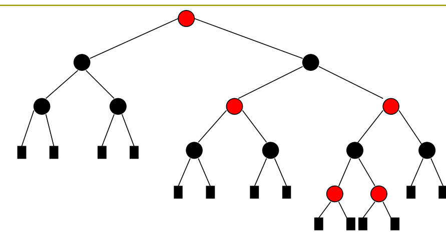
Insert (cont)



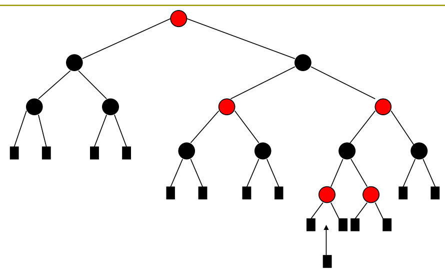
Use rotations



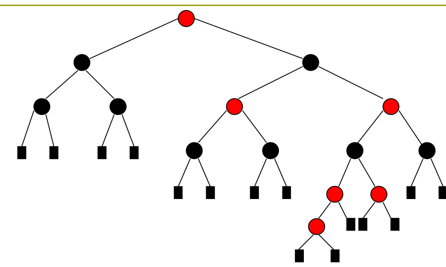
Insert (cont)



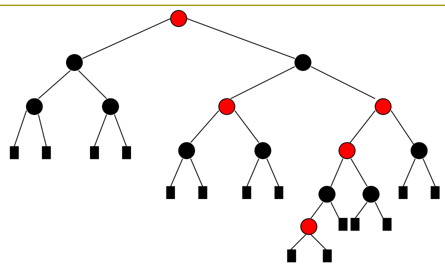
Insert (cont)



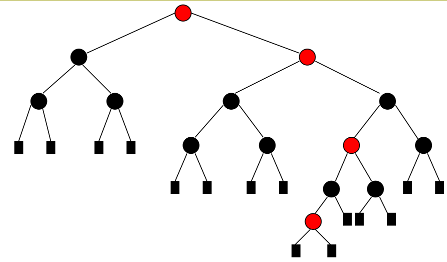
Insert (cont)



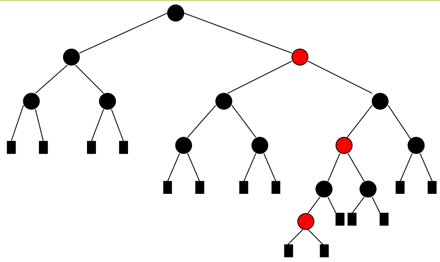
Insert (cont)



Insert (cont)



Insert (cont)

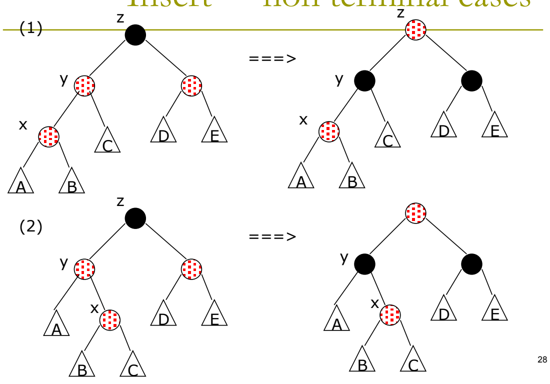


Insert -- definition

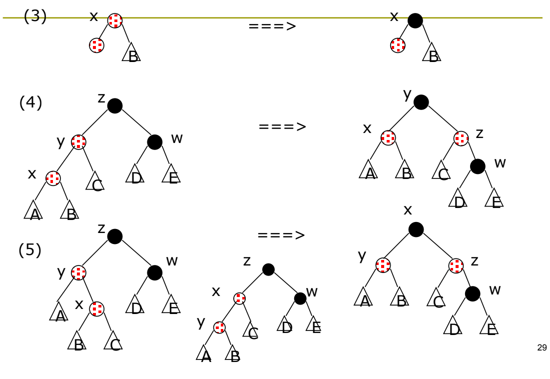
Convert a leaf to a red internal node with two leaves.

This may create violation to property 2. To restore it we walk up towards the root applying one of the following cases (each case has a symmetric version)

Insert -- non terminal cases



Insert -- terminal cases



Insert - analysis

O(log n) time worst case, since the height is O(log n)

Suppose you start with an empty tree and do m insertions such that the point of insertion is given to you each time, how much time does it take ?

Obviously O(mlog n),

but maybe we can prove it cannot be that bad ?

Insert - analysis

Each time we do a color-flip-step the number of red nodes decreases by one.

Φ(tree) = #red nodes

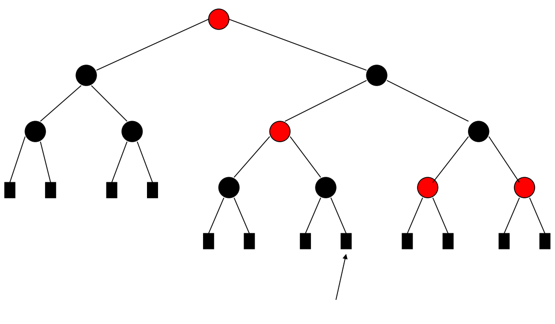
Actual(insert) = O(1) + #color-flips-steps

$\Delta\Phi$(insert) = O(1) - #color-flips-steps

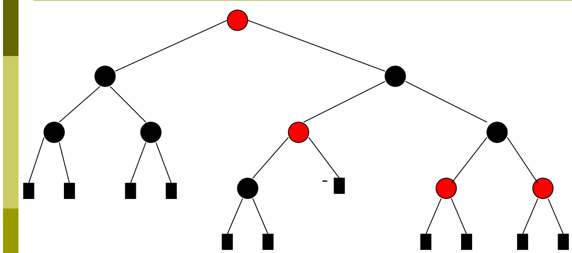
==> amortized(insert) = O(1)

and the sequence actually takes O(m) time.

Delete -- example

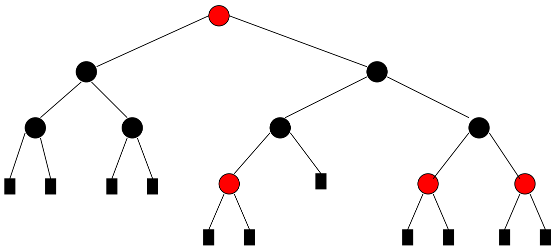


Delete -- example (cont)



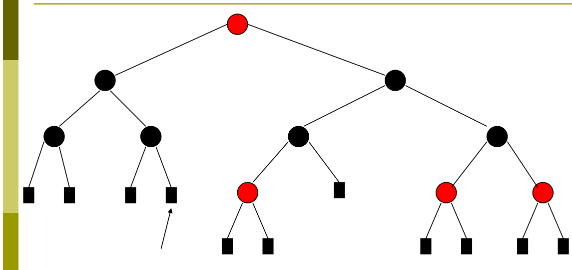
33

Delete -- example (cont)



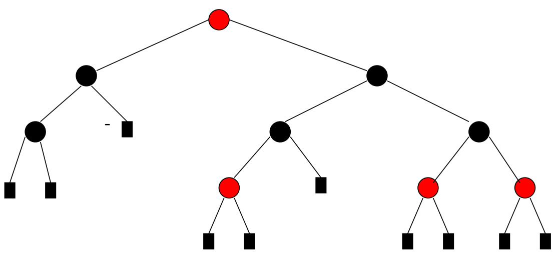
34

Delete -- example2 (cont)



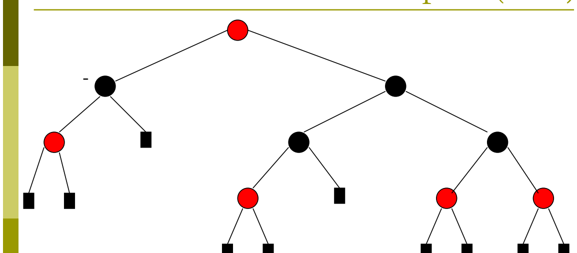
35

Delete -- example2 (cont)



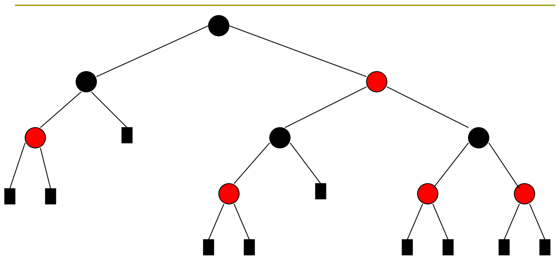
36

Delete -- example2 (cont)



37

Delete -- example2 (cont)



38

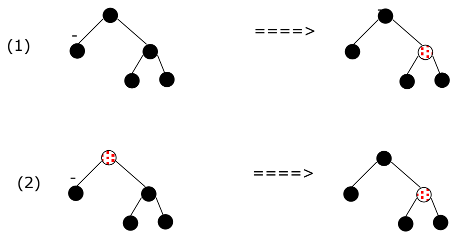
Delete -- definition

Replace the parent of the external node containing the item with the sibling subtree of the deleted item

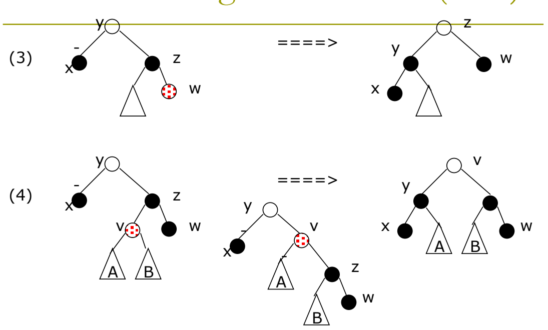
If the parent of the deleted item is black then we create a short node

To restore the black constraint we go bottom up applying one of the following cases.

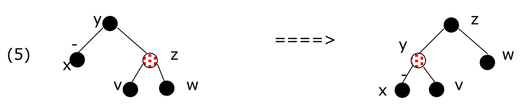
Delete -- fixing a short node



Delete -- fixing a short node (cont)



Delete -- fixing a short node (cont)



And apply one of the previous 3 cases.

Delete + insert -- analysis

O(log n) time, since the height is O(log n)

Suppose you start with an empty tree and do m insertions and deletions such that the point of insertion is given to you each time, how much time does it take ?

Obviously O(mlog n),

but maybe we can prove it cannot be that bad ?

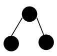
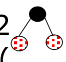
Delete + insert - analysis

The previous potential won't do the trick

Φ(tree) = #red nodes

Here are the transformation that we want to release potential

Delete + insert -- analysis

Φ(tree) = #() + 2#()

==> amortized(delete) = O(1)

amortized(insert) = O(1)

sequence of m delete and inserts, starting from an empty tree takes O(m) time

Delete + insert -- analysis