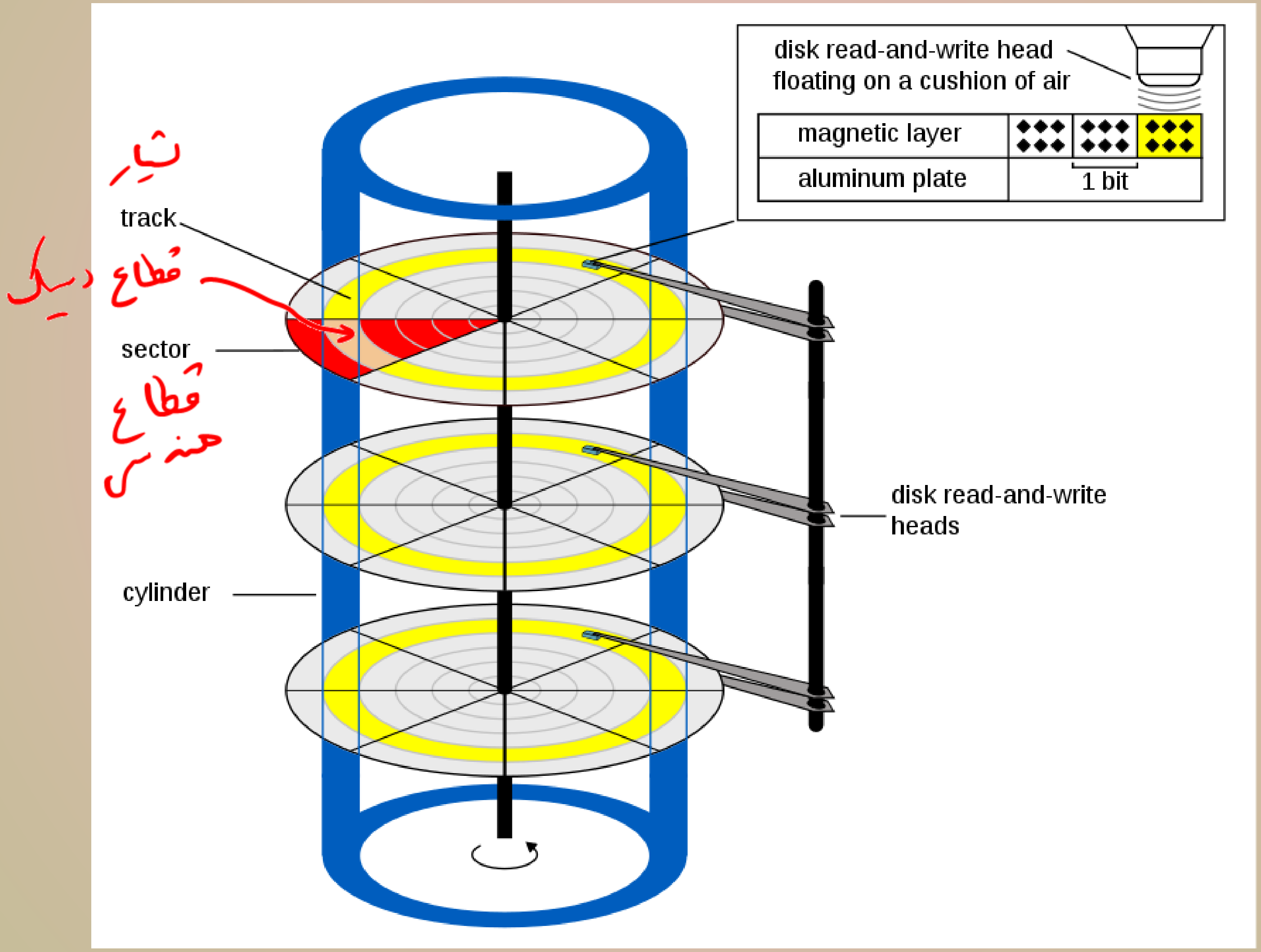
ساختمان داده ها

B-Trees

مدرس: غیاثی‌شیرازی
دانشگاه فردوسی مشهد

هندسه دیسک سخت



AVL Trees

$\log_2(n) \leqslant h \leqslant 1.4 \log_2(n)$

- $n = 2^{30} = 10^9$ (approx).
- 30 <= height <= 43.
- When the AVL tree resides on a disk, up to 43 disk access are made for a search.
- This takes up to (approx) 4 seconds.
- Not acceptable.

Red-Black Trees

$\log_2 n \leq h \leq 2 \log_2 n$

- $n = 2^{30} = 10^9$ (approx).
- 30 <= height <= 60.
- When the red-black tree resides on a disk, up to 60 disk access are made for a search.
- This takes up to (approx) 6 seconds.
- Not acceptable.

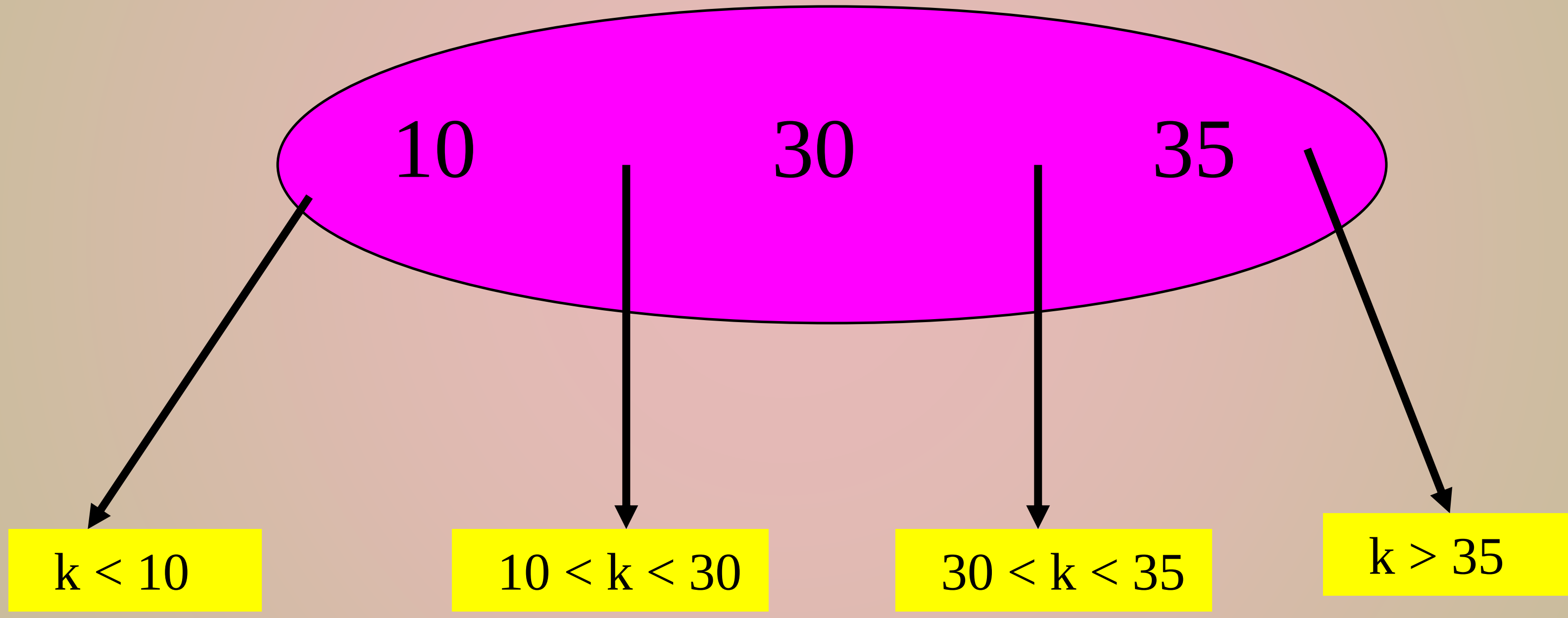
B-Trees

- Large degree B-trees used to represent very large dictionaries that reside on disk.
- Smaller degree B-trees used for internal-memory $\underbrace{\text{dictionaries}}_{\text{Map}}$ to overcome cache-miss penalties.

m-way Search Trees

- Each node has up to m – 1 pairs and m children.
- m = 2 => binary search tree.

4-Way Search Tree



Maximum # Of Pairs

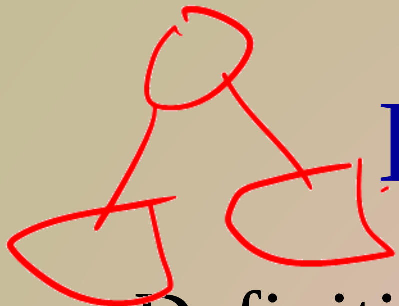
<Key, Value>

- Happens when all internal nodes are m-nodes.
- Full degree m tree.
- # of nodes $= 1 + m + m^2 + m^3 + \ldots + m^{h-1}$ $= (m^h - 1)/(m - 1)$.
- Each node has $m - 1$ pairs.
- So, # of pairs $= m^h - 1$.

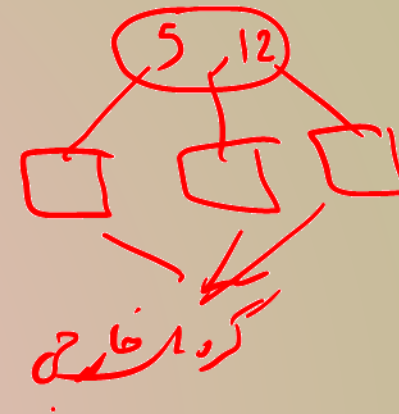
h

Capacity Of m-Way Search Tree

	m = 2	m = 200
h = 3	7	$8 * 10^{6} - 1$
h = 5	31	$3.2 * 10^{11} - 1$
h = 7	127	$1.28 * 10^{16} - 1$



Definition Of B-Tree



- Definition assumes external nodes (extended m-way search tree).
- B-tree of order m.
  - m-way search tree. می‌تواند گره‌ی خارجی باشد
  - Not empty => root has at least 2 children.
  - Remaining internal nodes (if any) have at least ceil(m/2) children. عنصر دارد $\lceil m/2 \rceil - 1$ فرزند $\lceil m/2 \rceil$
  - External (or failure) nodes on same level.

$m=3$ $m=4$

2-3 And 2-3-4 Trees

$\lceil 3/2 \rceil = 2$

$\lceil 4/2 \rceil = 2$ حداقل تعداد فرزند

- B-tree of order m.
  - m-way search tree.
  - Not empty => root has at least 2 children.
  - Remaining internal nodes (if any) have at least ceil(m/2) children. $\lceil m/2 \rceil = 2$
  - External (or failure) nodes on same level.

به غیر از ریشه

فرق بین ریشه و بقیه گره‌ها است

- 2-3 tree is B-tree of order 3.
- 2-3-4 tree is B-tree of order 4.

B-Trees Of Order 5 And 2

- B-tree of order m. $\lceil 5/2 \rceil = 3$
  - m-way search tree.
  - Not empty => root has at least 2 children.
  - Remaining internal nodes (if any) have at least ceil(m/2) children.
  - External (or failure) nodes on same level.

- B-tree of order 5 is 3-4-5 tree (root may be 2-node though).
- B-tree of order 2 is full binary tree. $h \Rightarrow 2^h - 1$ گره

Minimum # Of Pairs

- n = # of pairs.
- # of external nodes = n + 1.
- Height = h => external nodes on level h + 1.

level	# of nodes
1	1
2	>= 2
3	>= 2*ceil(m/2)
h + 1	>= 2*ceil(m/2)$^{h-1}$

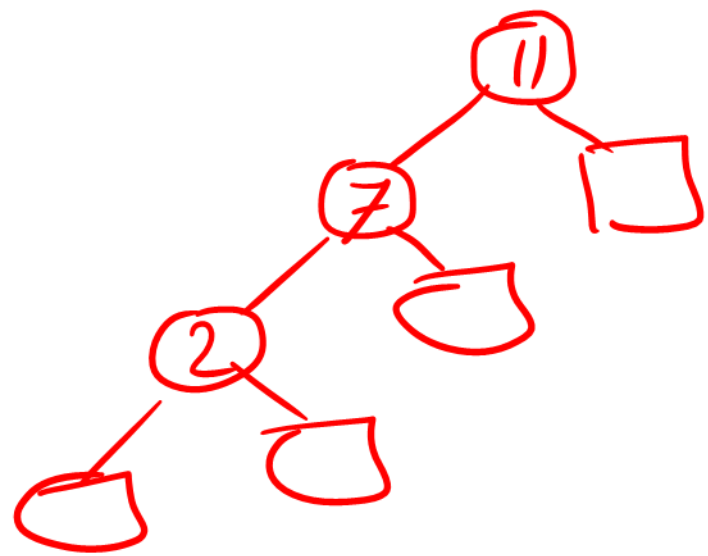
تعداد گره‌های خارجی

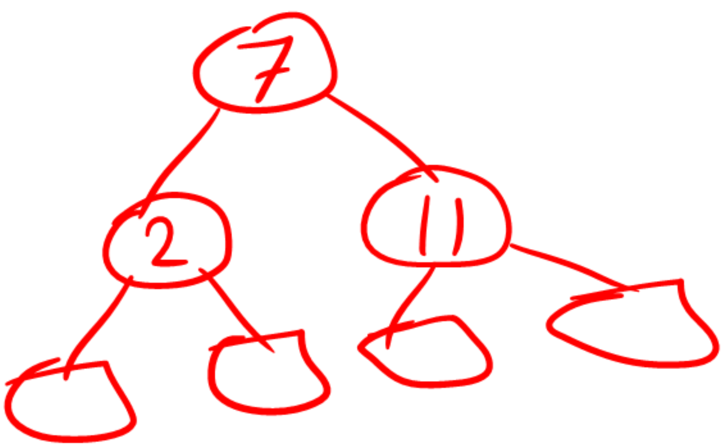
$n + 1 >= 2*ceil(m/2)^{h-1}$, $h >= 1$

2 7 11

n تعداد داده‌ها

n+1 گره خارج





Minimum # Of Pairs

$n + 1 >= 2*\text{ceil}(m/2)^{h-1}$, $h >= 1$

$\frac{n+1}{2} \geq \lceil m/2 \rceil^{h-1}$

$h-1 \leq \log_{\lceil m/2 \rceil}\left(\frac{n+1}{2}\right)$

$h \leq \log_{\lceil m/2 \rceil}\left(\frac{n+1}{2}\right) + 1$

- m = 200.

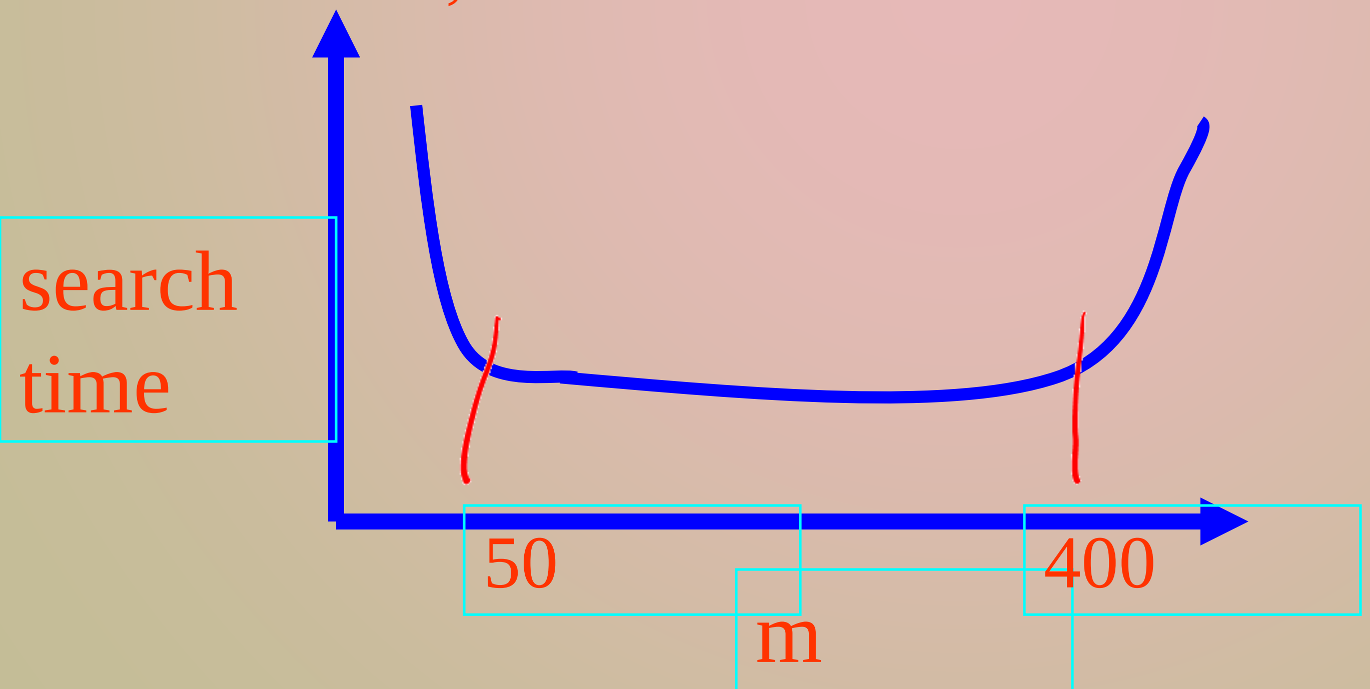
height	# of pairs
2	>= 199
3	>= 19,999
4	>= $2 * 10^6 - 1$
5	>= $2 * 10^8 - 1$

$h <= \log_{\text{ceil}(m/2)} [(n+1)/2] + 1$

Choice Of m

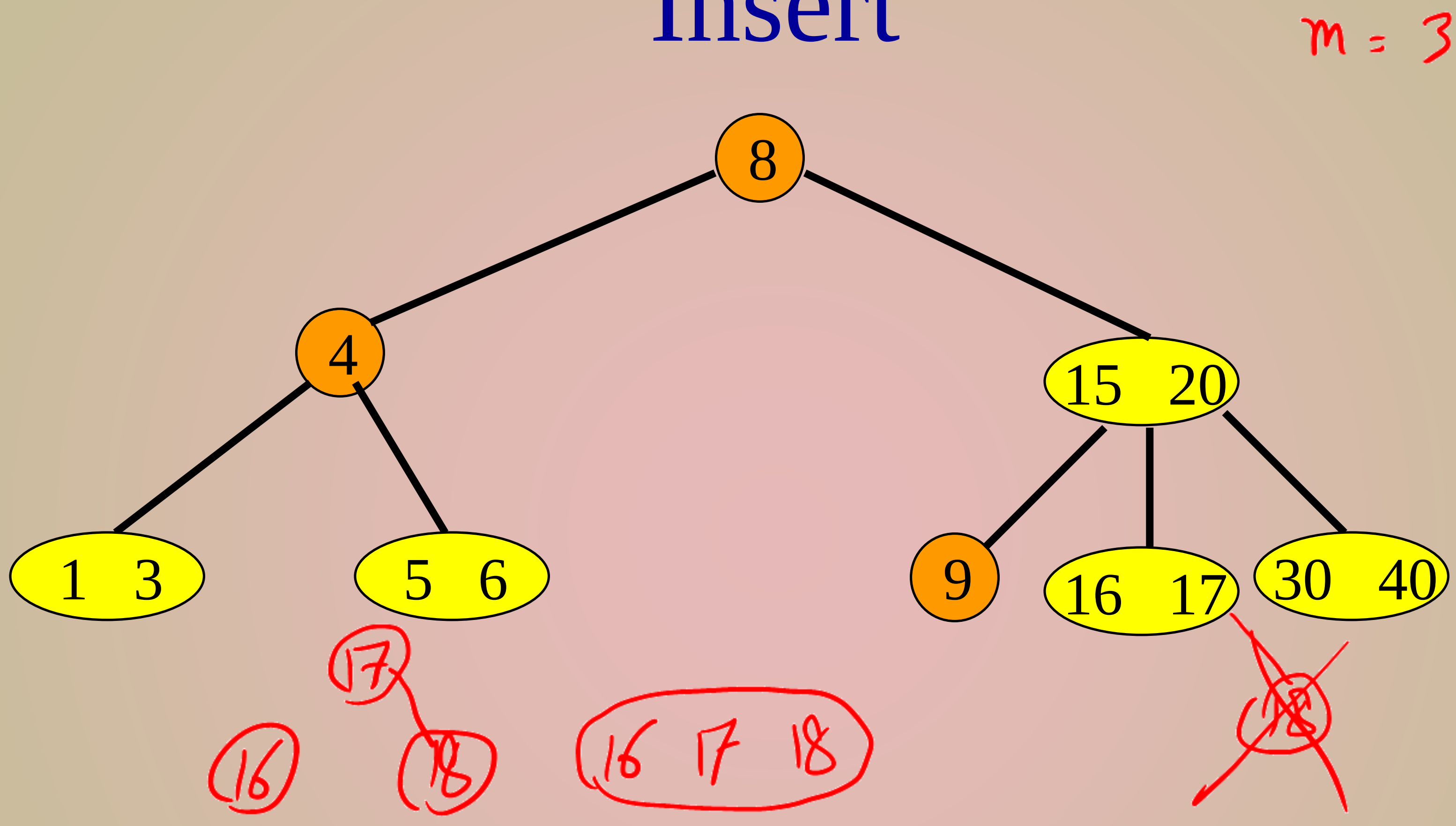
- Worst-case search time.
  - (time to fetch a node + time to search node) * height
  - $(a + b*m + c * \log_2 m) * h$

  where a, b and c are constants.



انتخاب m
به پارامترهای
دیسک خیلی
حساس نیست

Insert



Insertion into a full leaf triggers bottom-up node splitting pass.

Split An Overfull Node

زوج

$m \mid a_0\ p_1\ a_1\ p_2\ a_2 \ldots p_m\ a_m$

زیردرخت

- a_i is a pointer to a subtree.
- p_i is a dictionary pair.

$P_{\lceil m/2 \rceil}$

گره فعلی

$P_1, \ldots, P_{\lceil m/2 \rceil -1}$

جدید

$P_{\lceil m/2 \rceil +1}, \ldots, P_m$

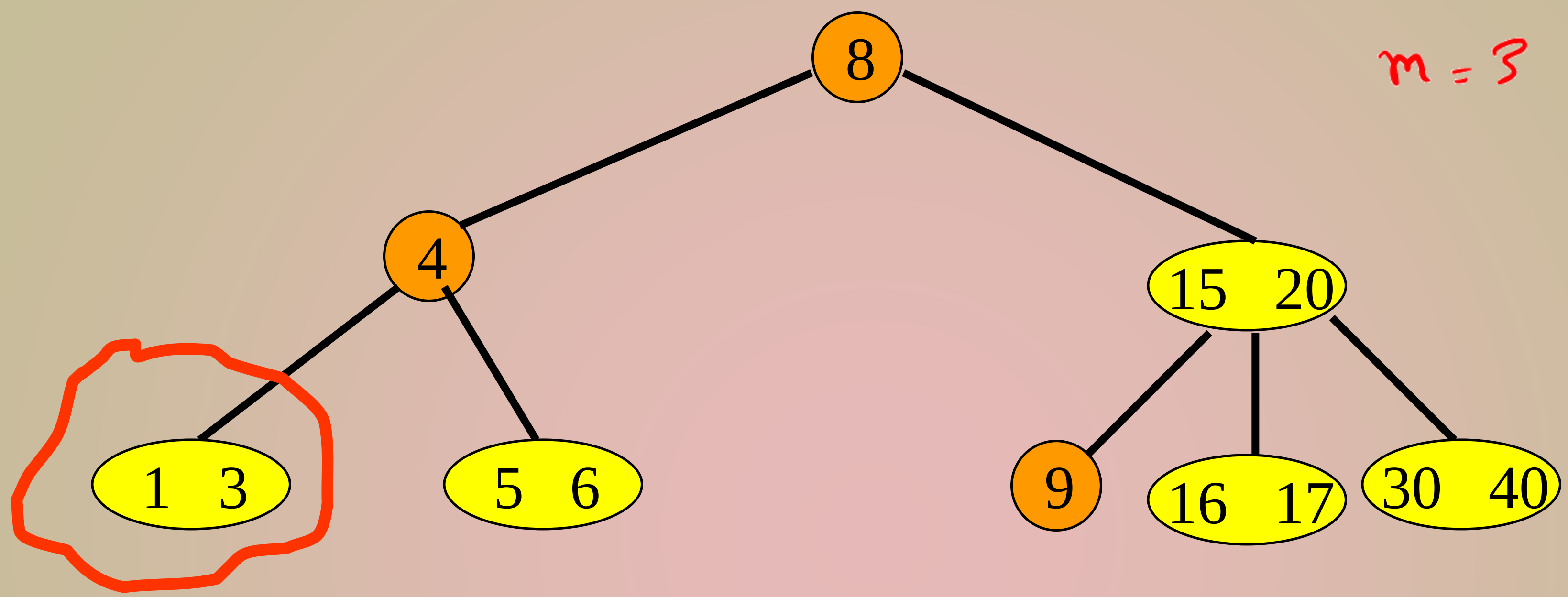
$\text{ceil}(m/2)-1\ a_0\ p_1\ a_1\ p_2\ a_2 \ldots p_{\text{ceil}(m/2)-1}\ a_{\text{ceil}(m/2)-1}$

$m-\text{ceil}(m/2)\ a_{\text{ceil}(m/2)}\ p_{\text{ceil}(m/2)+1}\ a_{\text{ceil}(m/2)+1} \ldots p_m\ a_m$

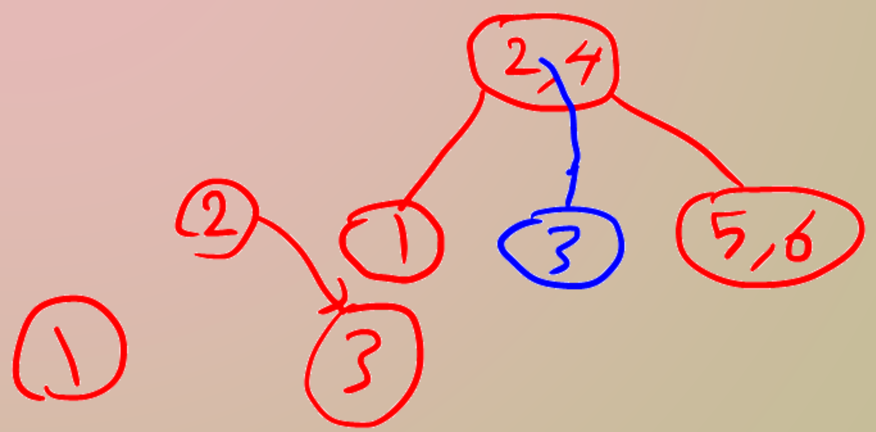
- $p_{\text{ceil}(m/2)}$ plus pointer to new node is inserted in parent.

Insert

m = 3

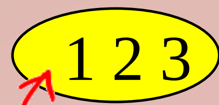


- Insert a pair with key = 2.
- New pair goes into a 3-node.

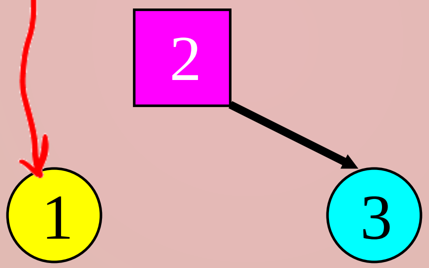


Insert Into A Leaf 3-node

- Insert new pair so that the 3 keys are in ascending order.

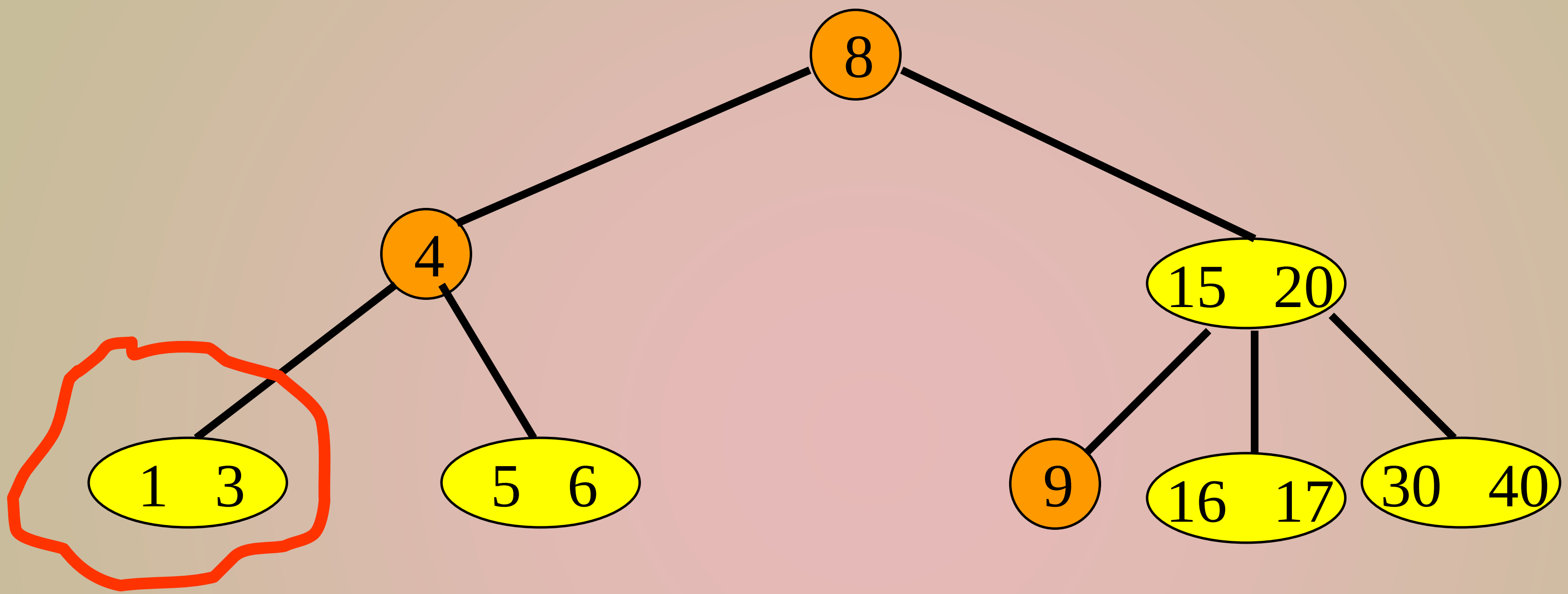


- Split overflowed node around middle key.



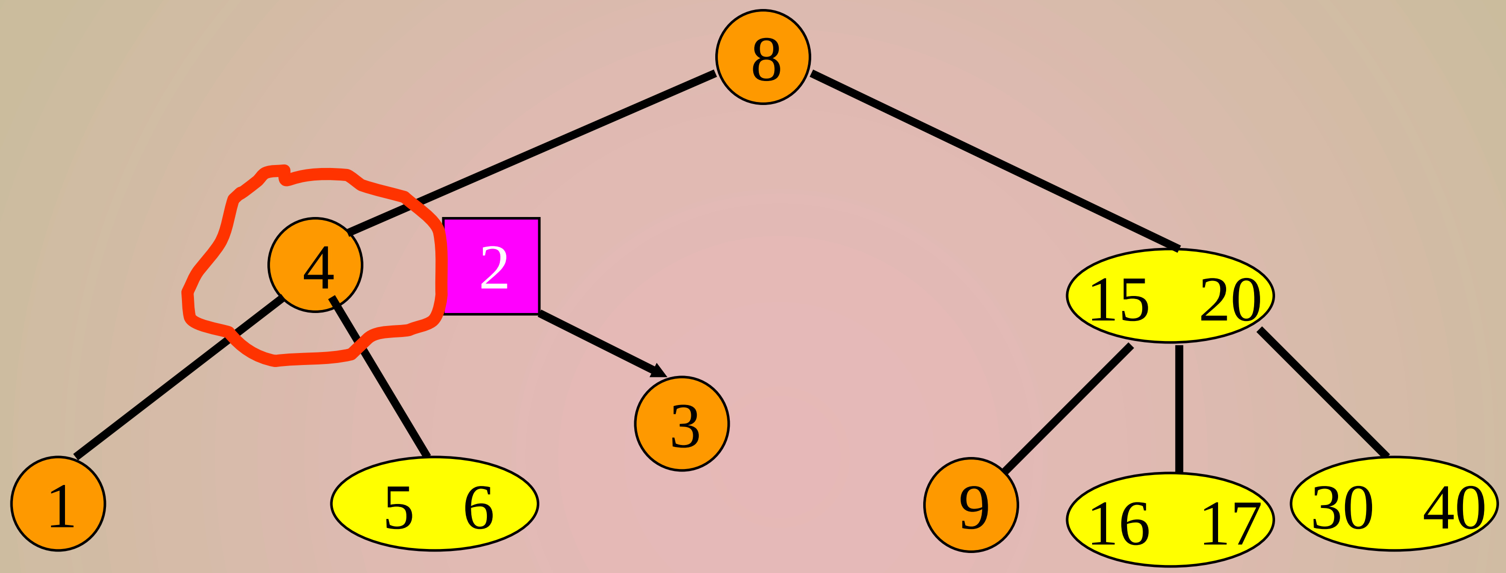
- Insert middle key and pointer to new node into parent.

Insert



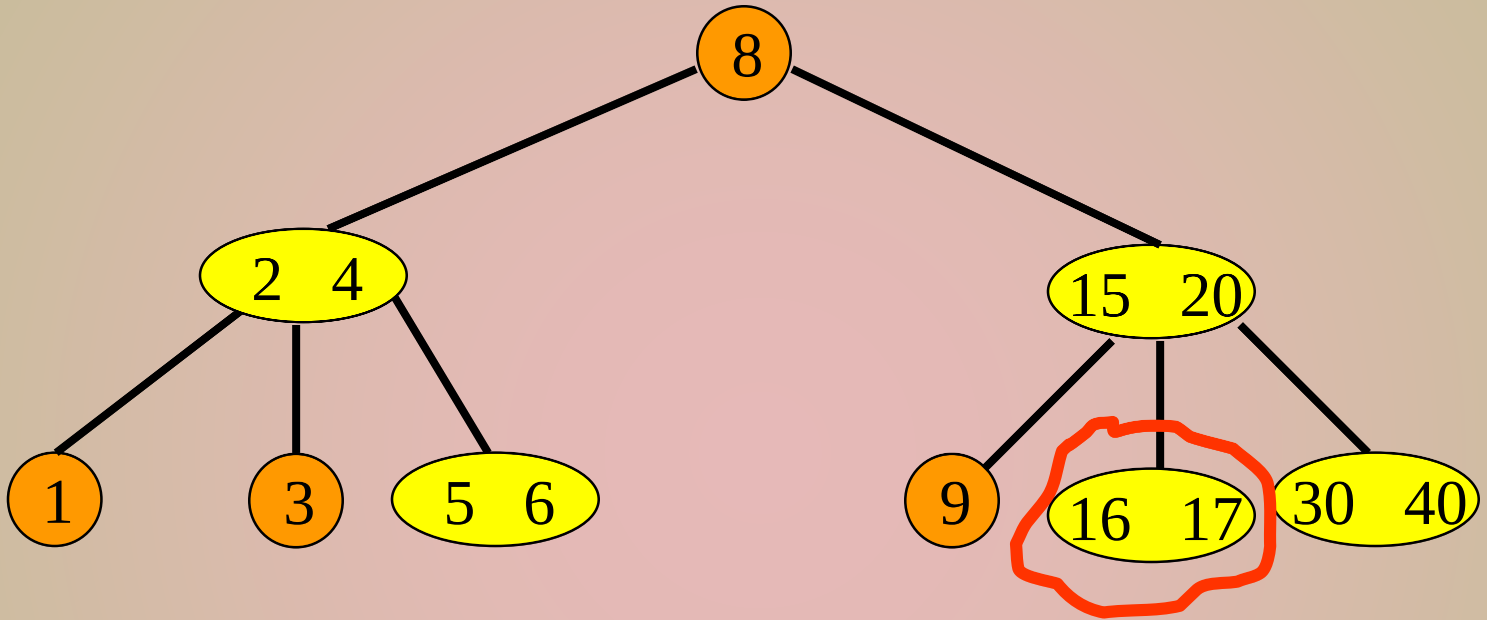
- Insert a pair with key = 2.

Insert

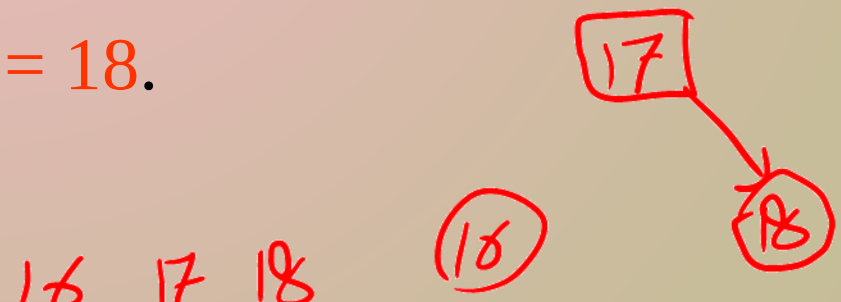


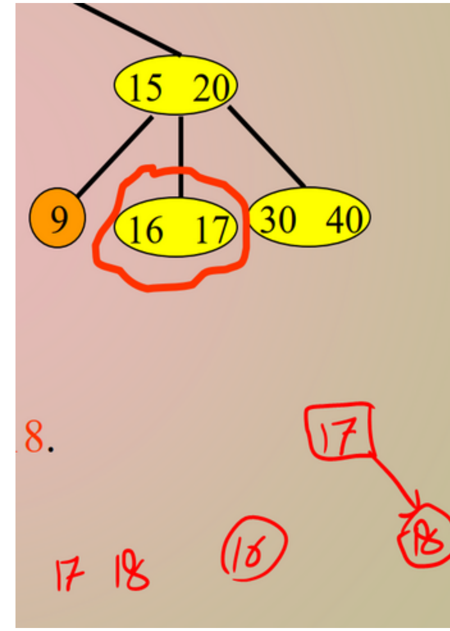
- Insert a pair with key = 2 plus a pointer into parent.

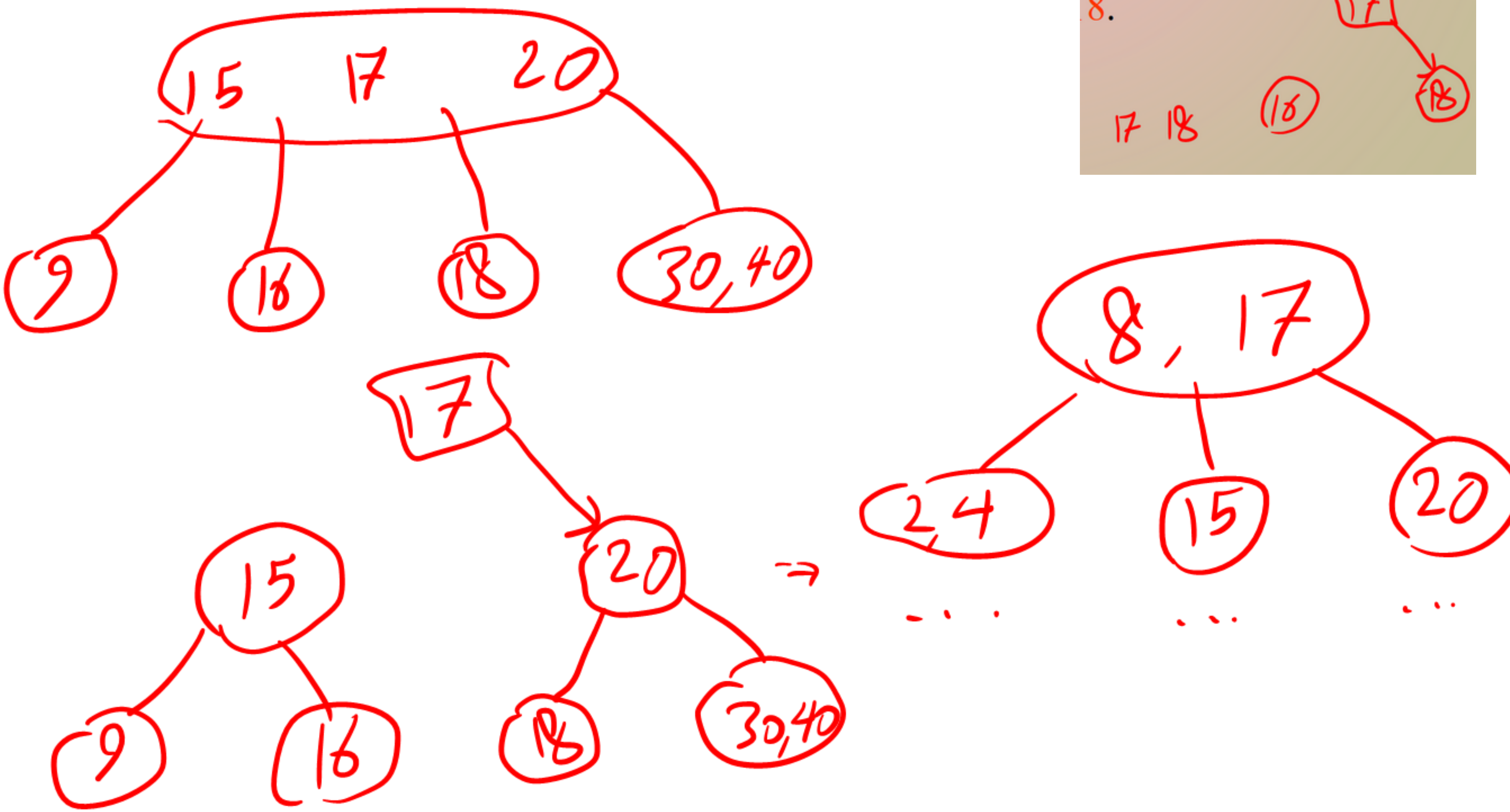
Insert



- Now, insert a pair with key = 18.

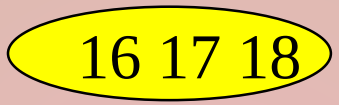



15 20
9
16 17
30 40
8.
17
16
17 18
18

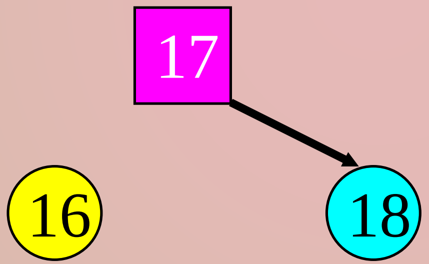

15 17 20
9
16
18
30,40
17
15
20
8, 17
2,4
15
20

Insert Into A Leaf 3-node

- Insert new pair so that the 3 keys are in ascending order.

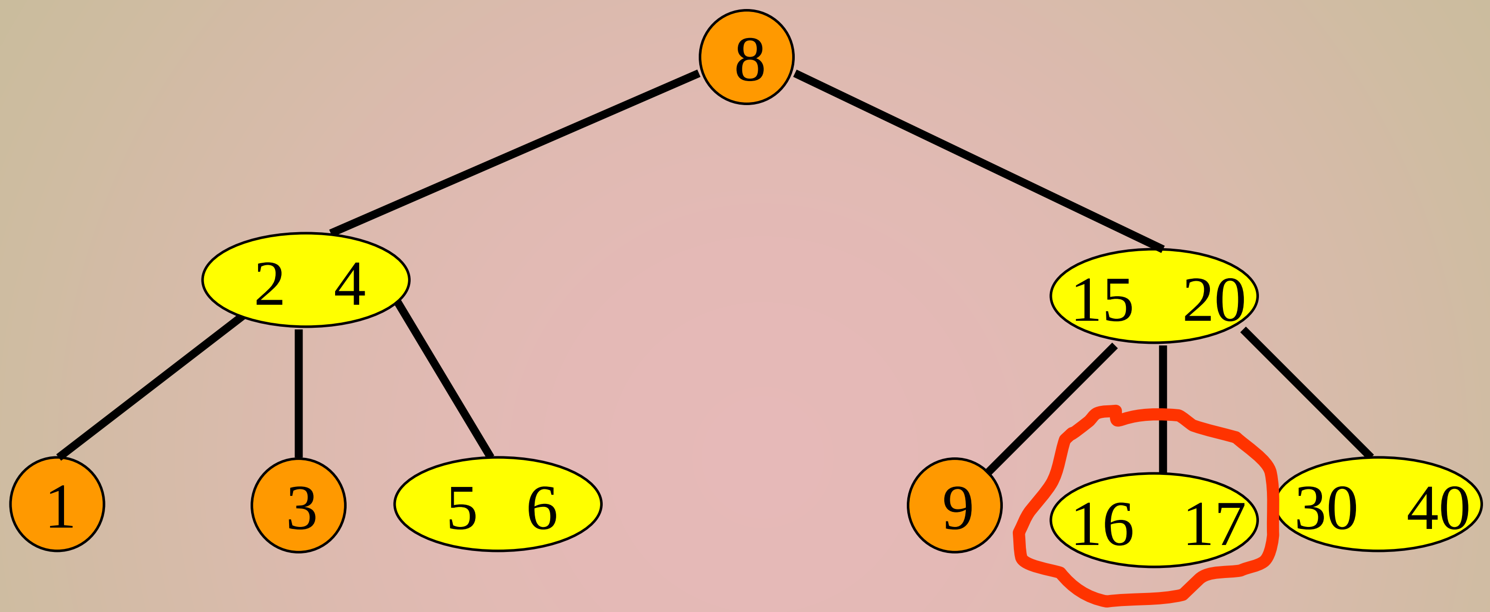


- Split the overflowed node.



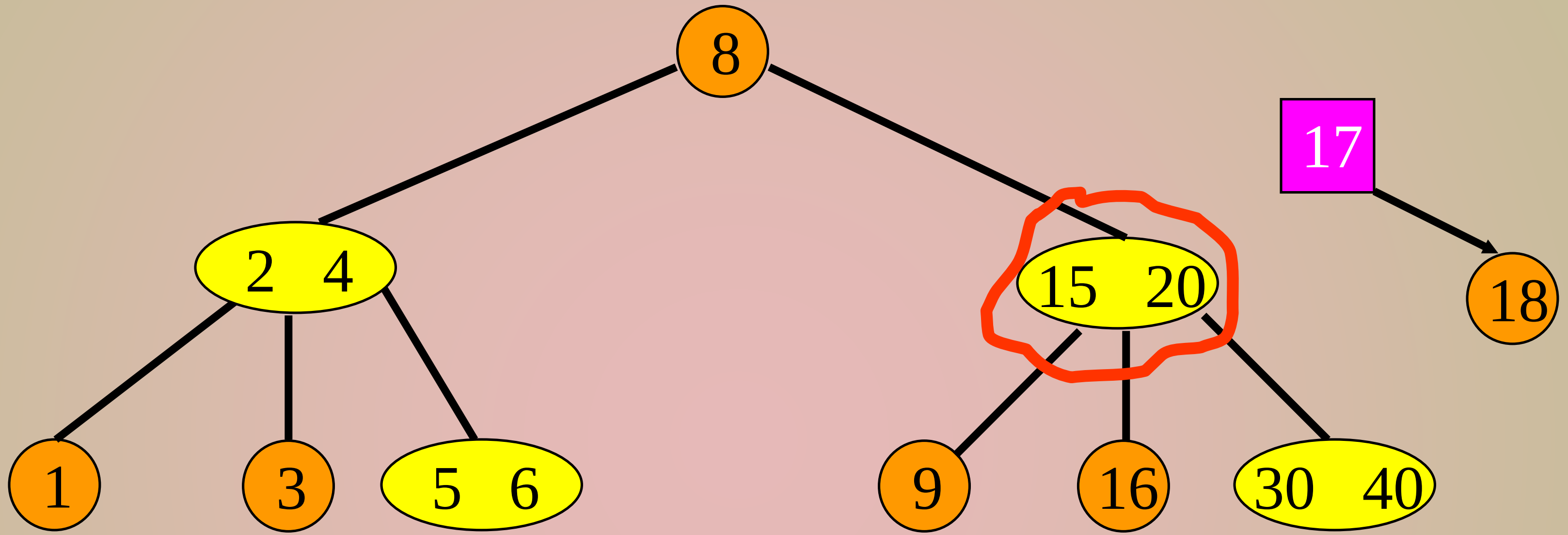
- Insert middle key and pointer to new node into parent.

Insert



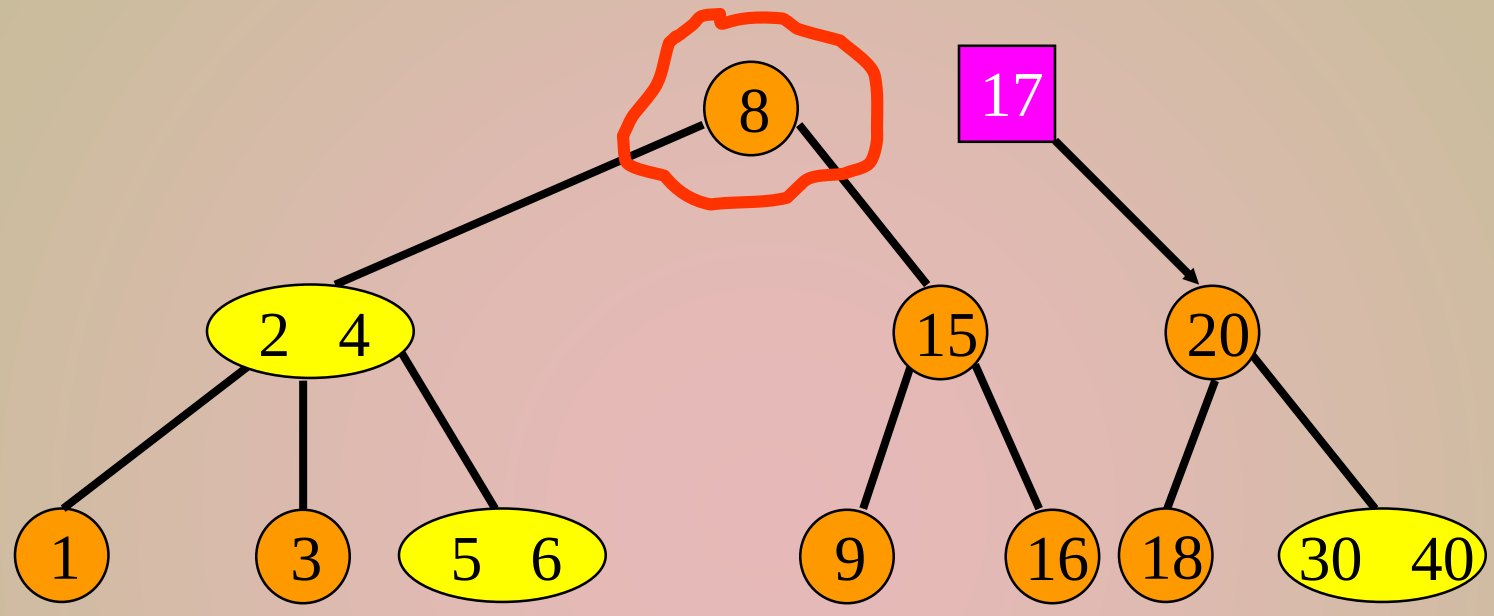
- Insert a pair with key = 18.

Insert



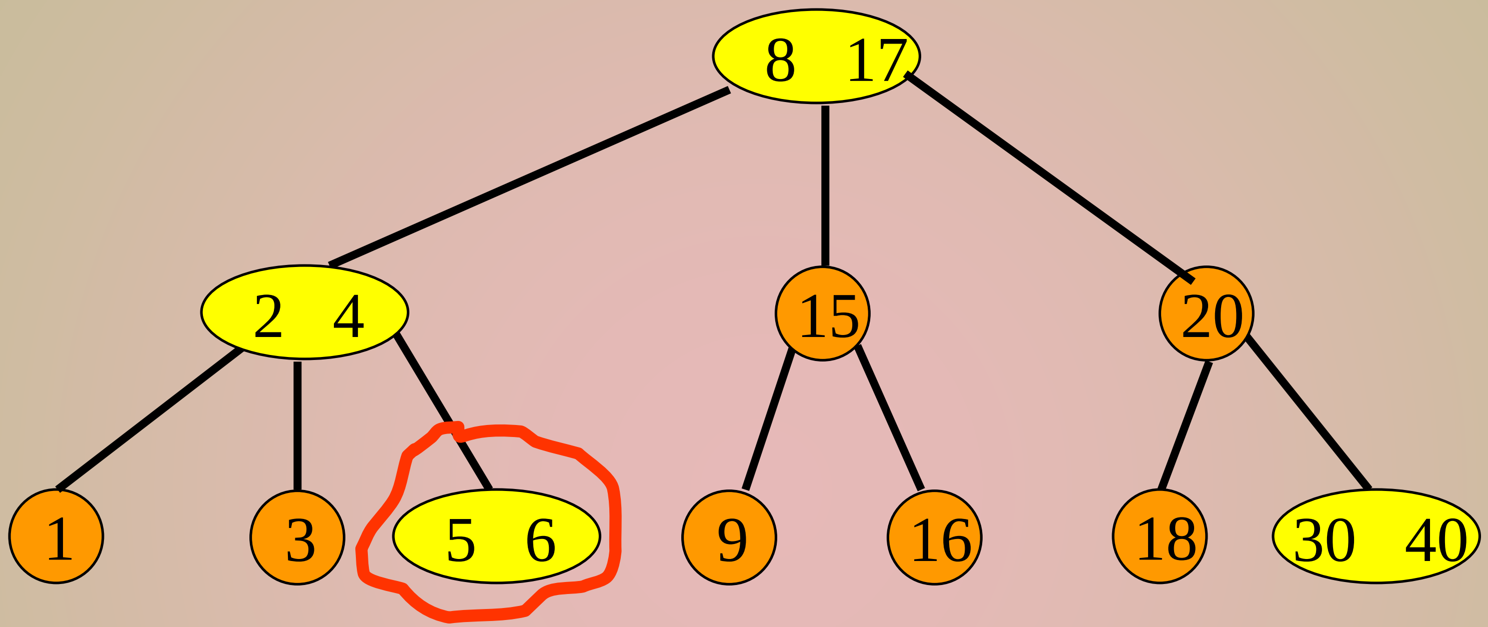
- Insert a pair with key = 17 plus a pointer into parent.

Insert



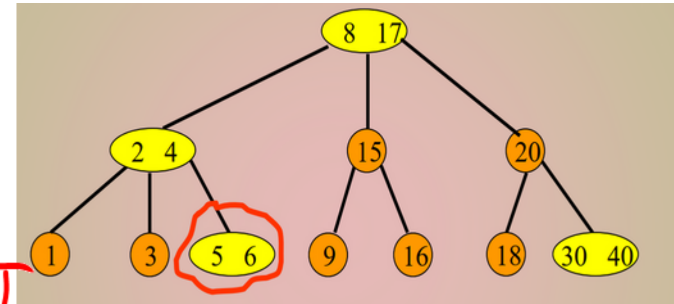
- Insert a pair with key = 17 plus a pointer into parent.

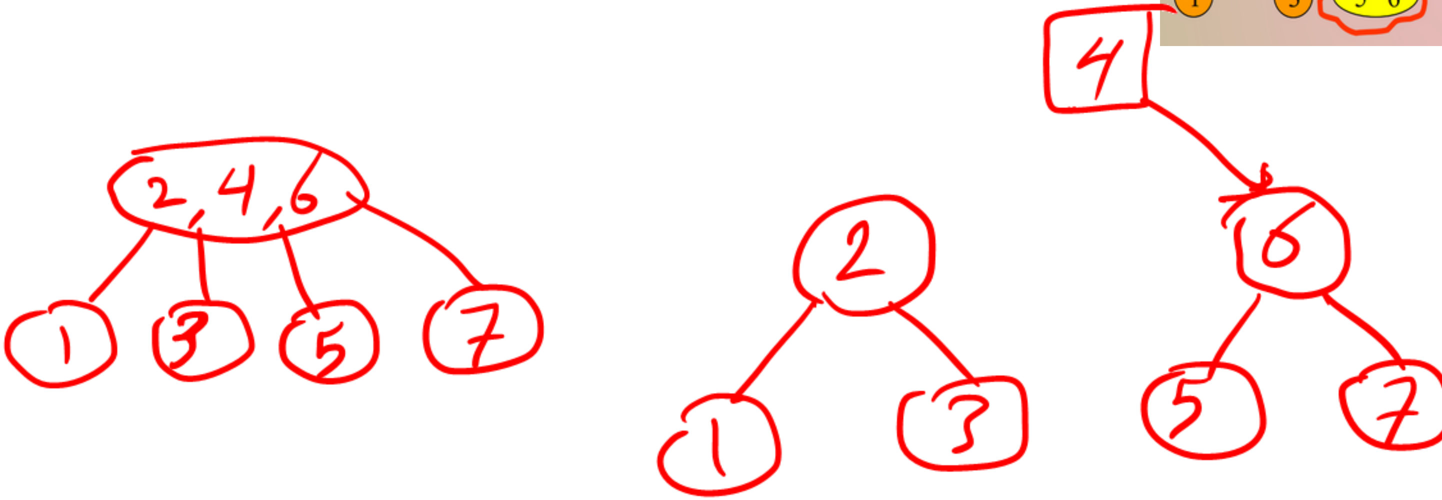
Insert

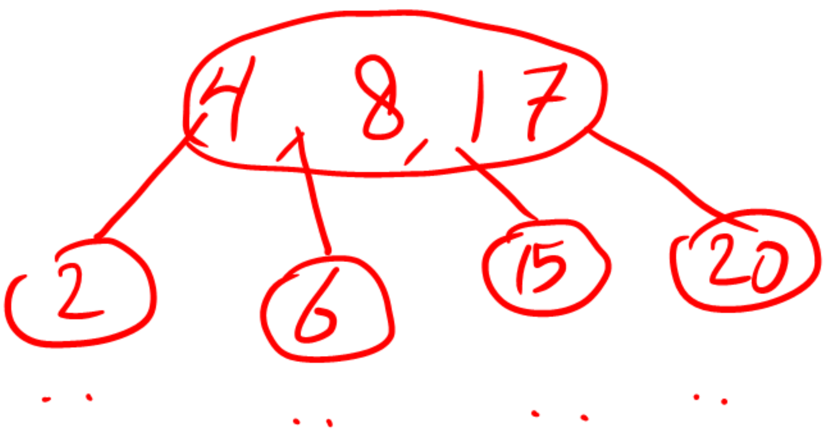


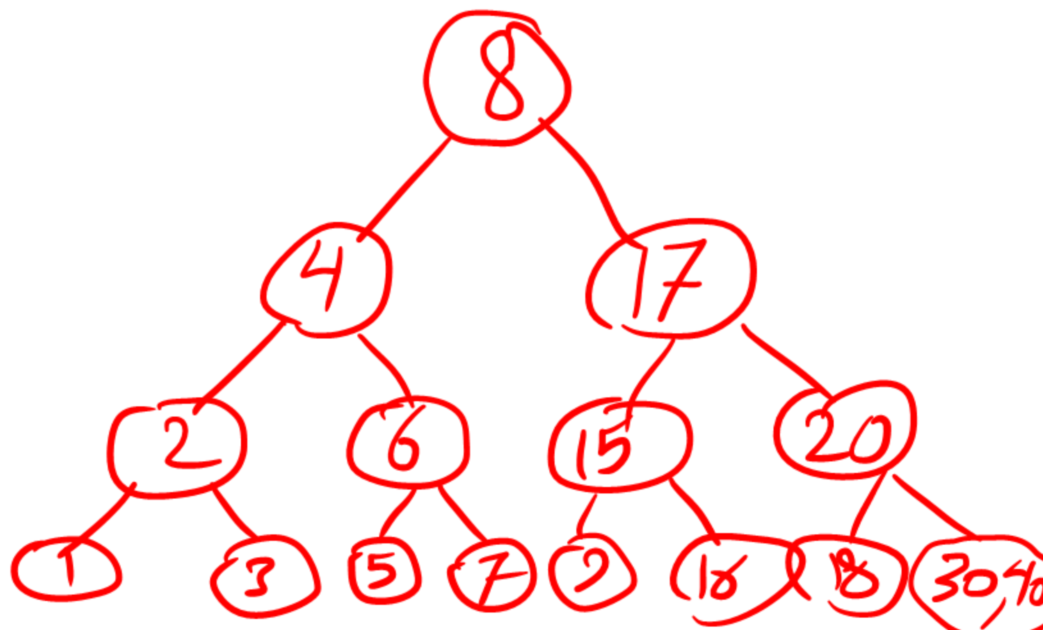
- Now, insert a pair with key = 7.


5,6,7
5
6
7

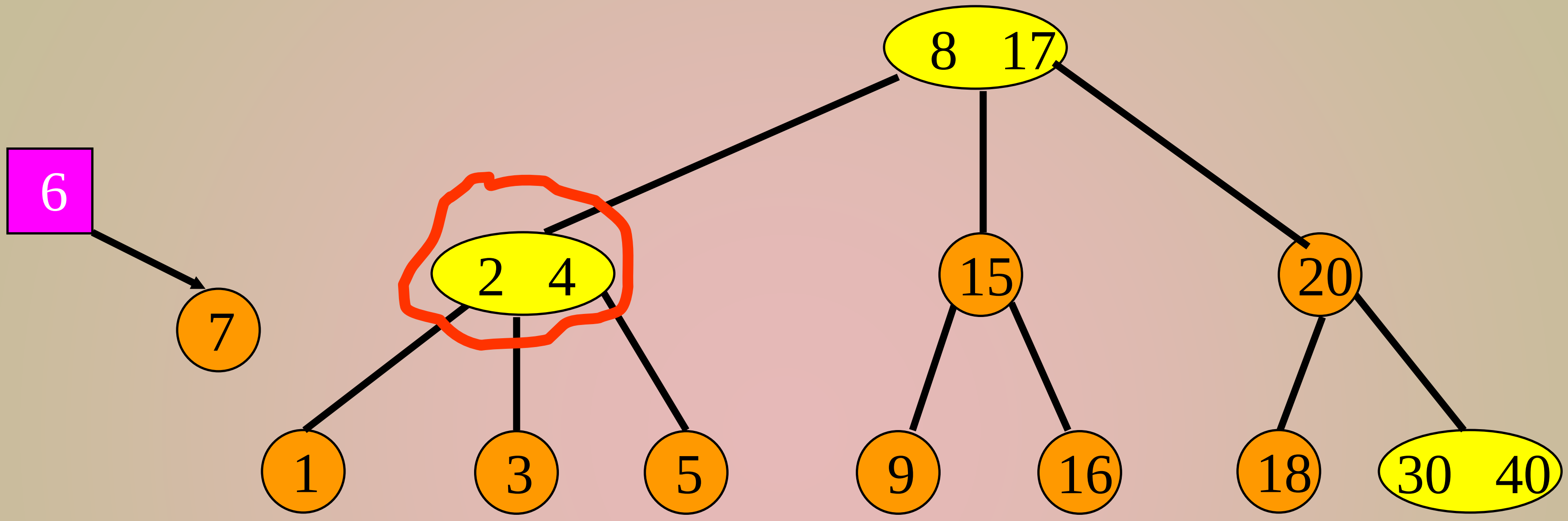

8 17
2 4
15
20
1
3
5 6
9
16
18
30 40


2,4,6
1
3
5
7
2
1
3
4
6
5
7


4, 8,17
2
6
15
20

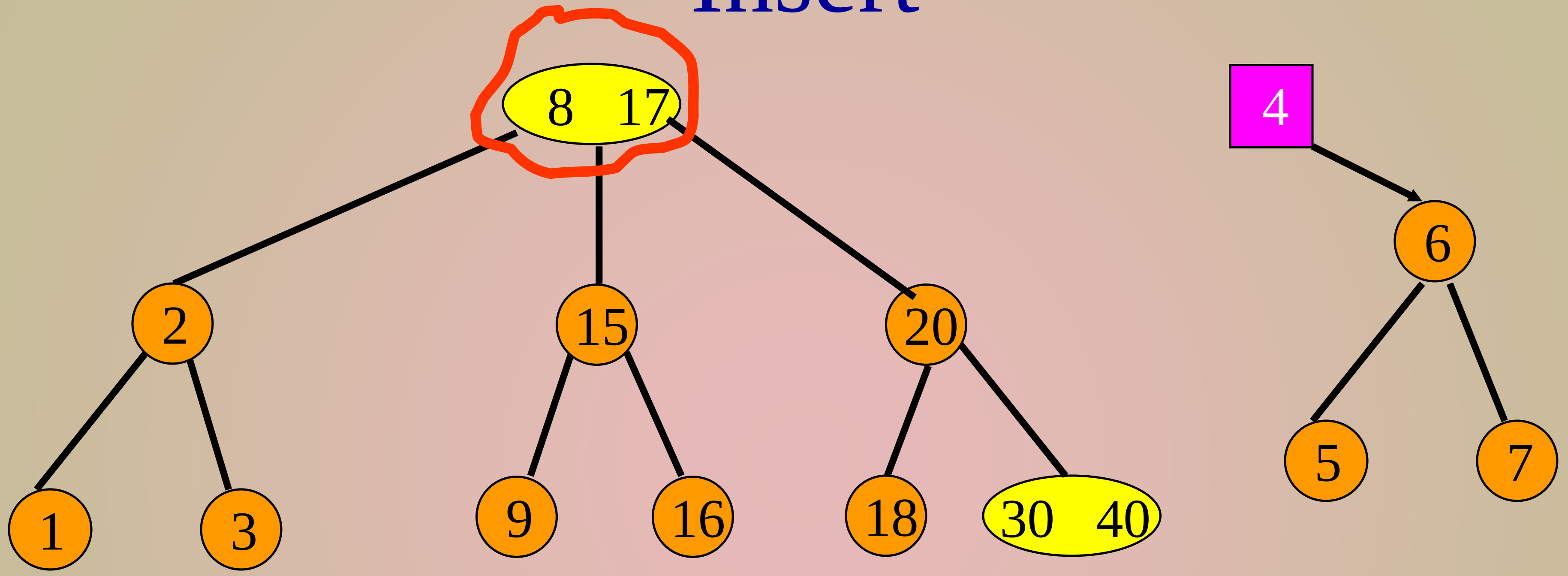

8
4
17
2
6
15
20
1
3
5
7
9
16
18
30,40

Insert



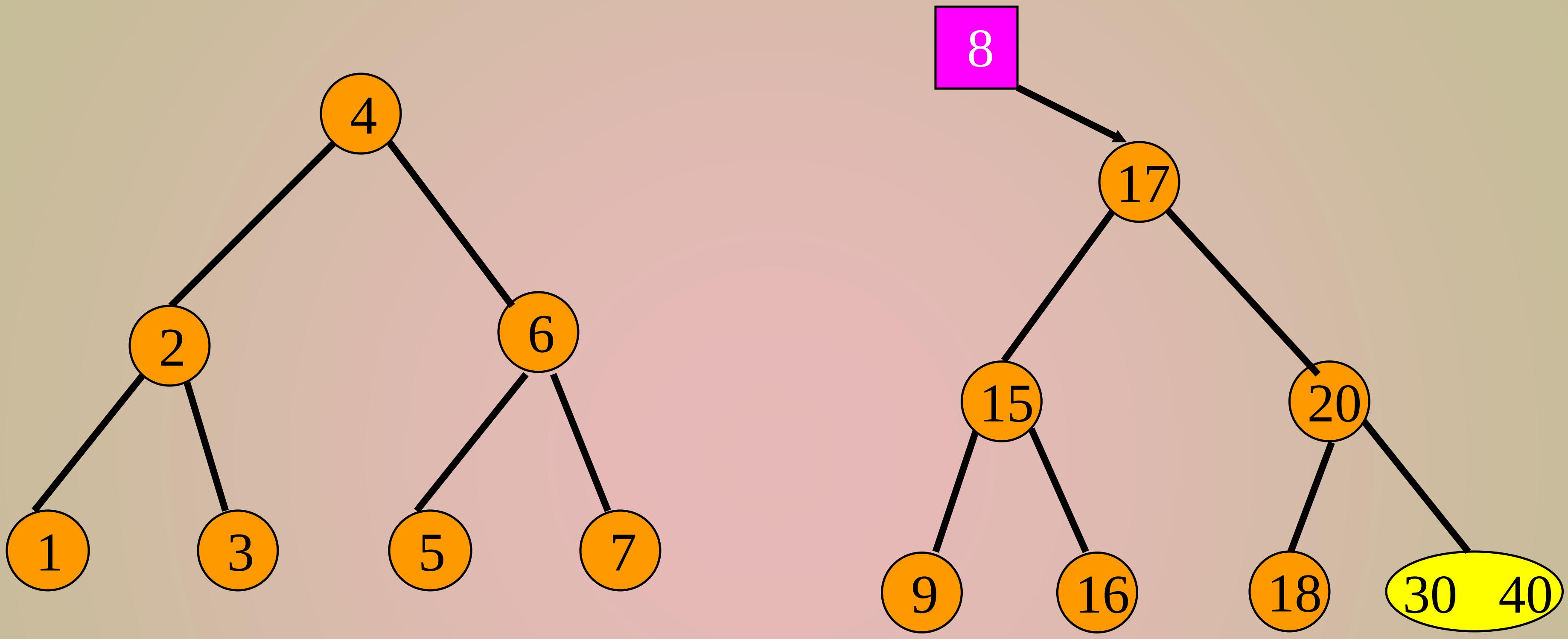
- Insert a pair with key = 6 plus a pointer into parent.

Insert



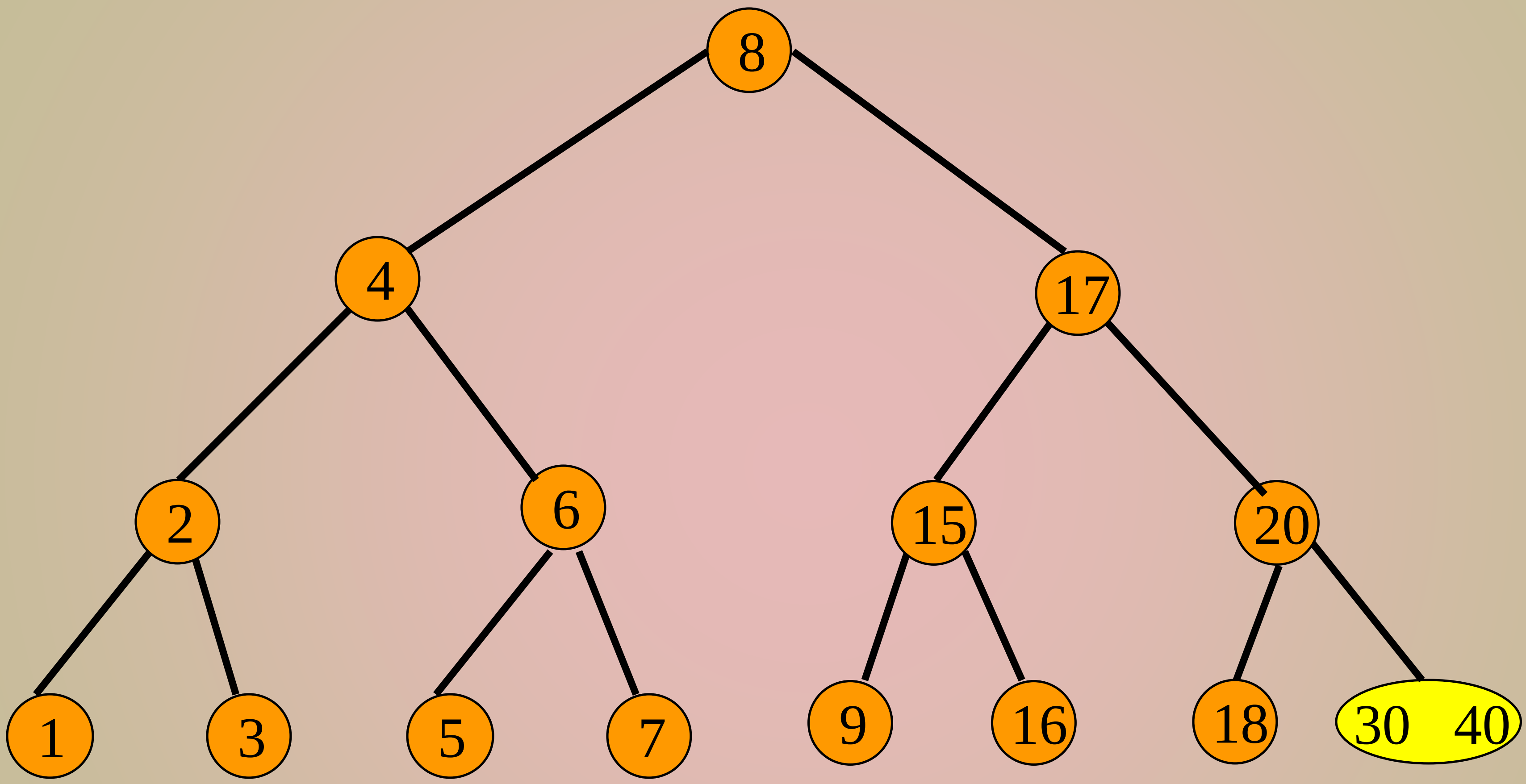
- Insert a pair with key = 4 plus a pointer into parent.

Insert



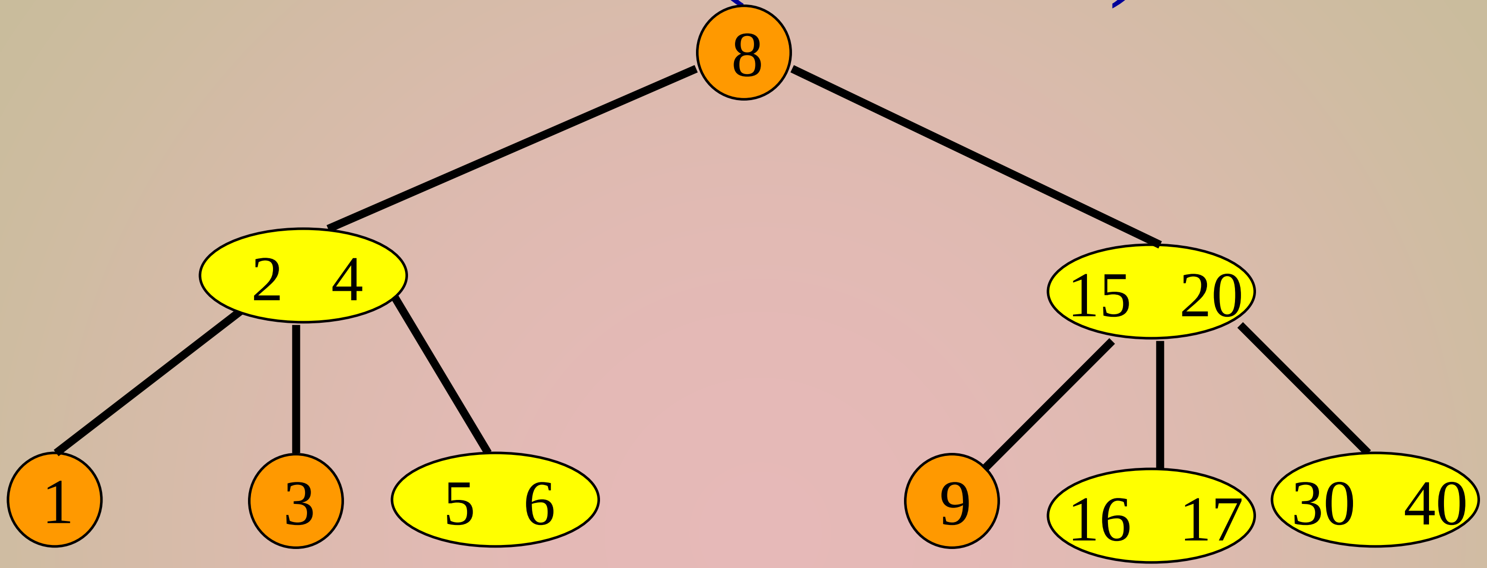
- Insert a pair with key = 8 plus a pointer into parent.
- There is no parent. So, create a new root.

Insert



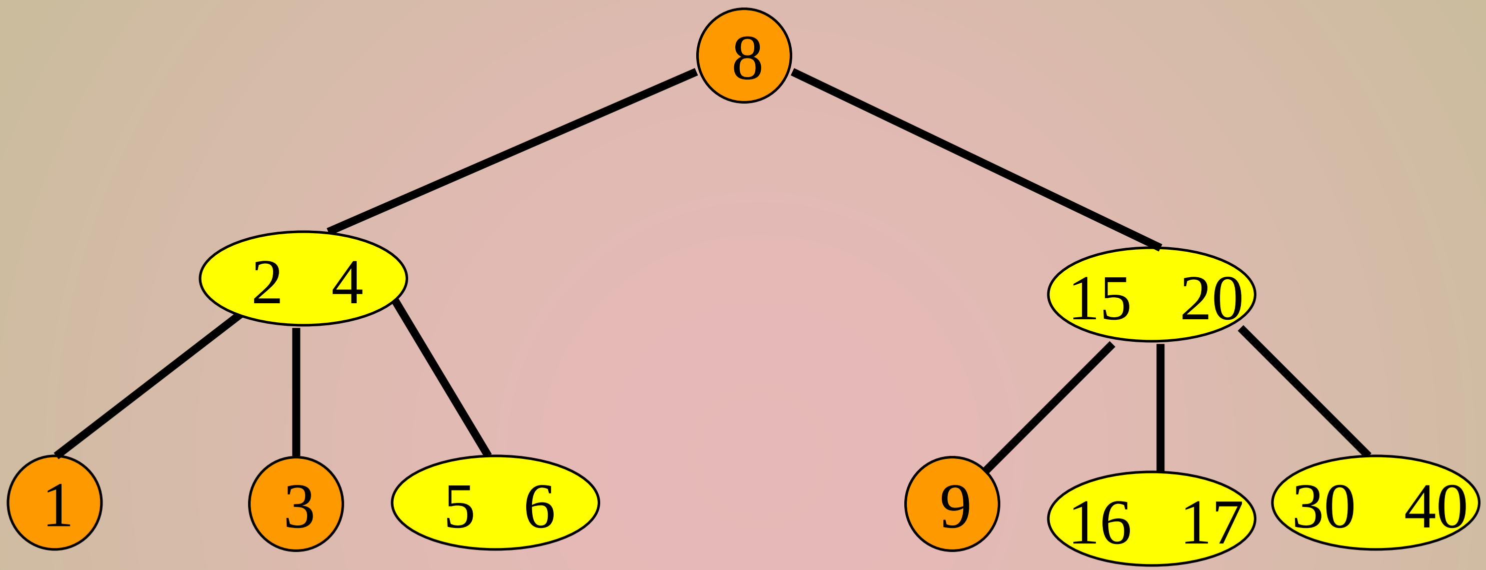
- Height increases by 1.

Delete (2-3 tree)



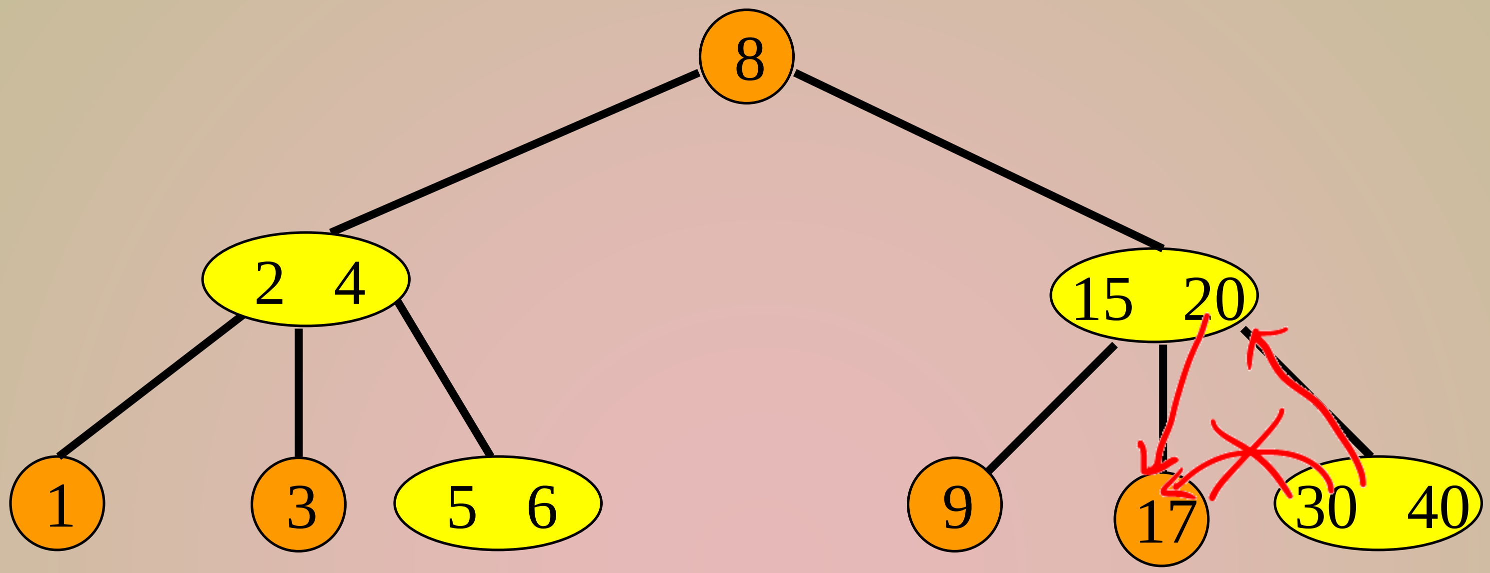
- Delete the pair with key = 8.
- Transform deletion from interior into deletion from a leaf.
- Replace by largest in left subtree.

Delete From A Leaf



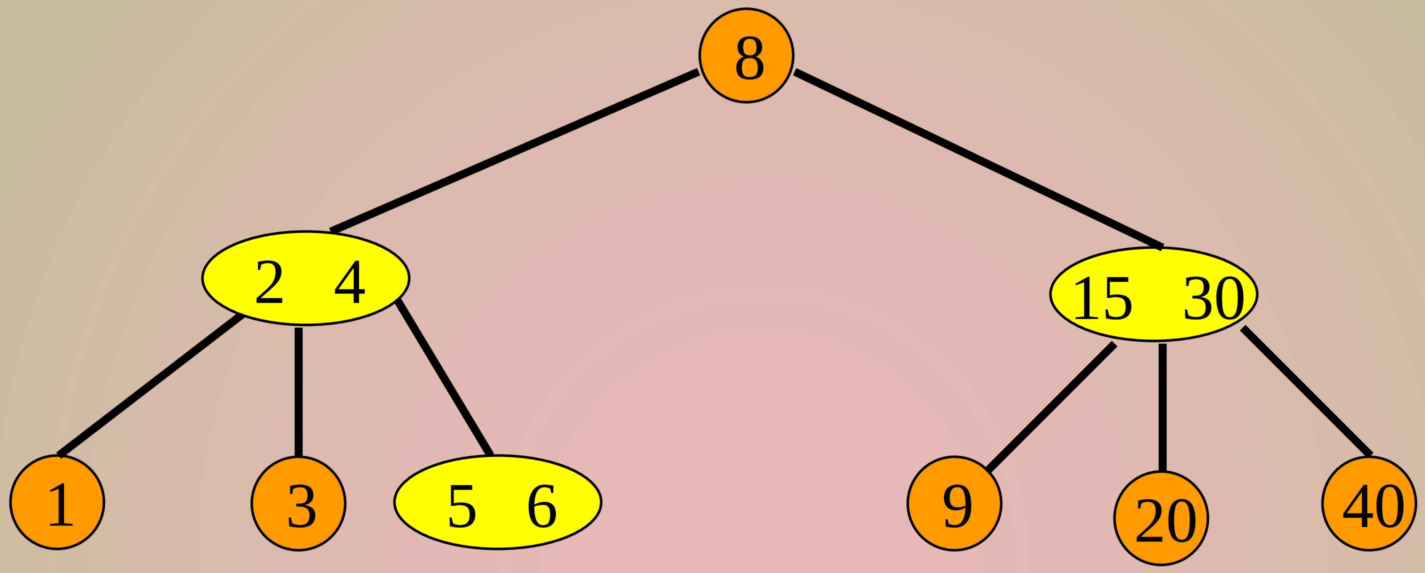
- Delete the pair with key = 16.
- 3-node becomes 2-node.

Delete From A Leaf



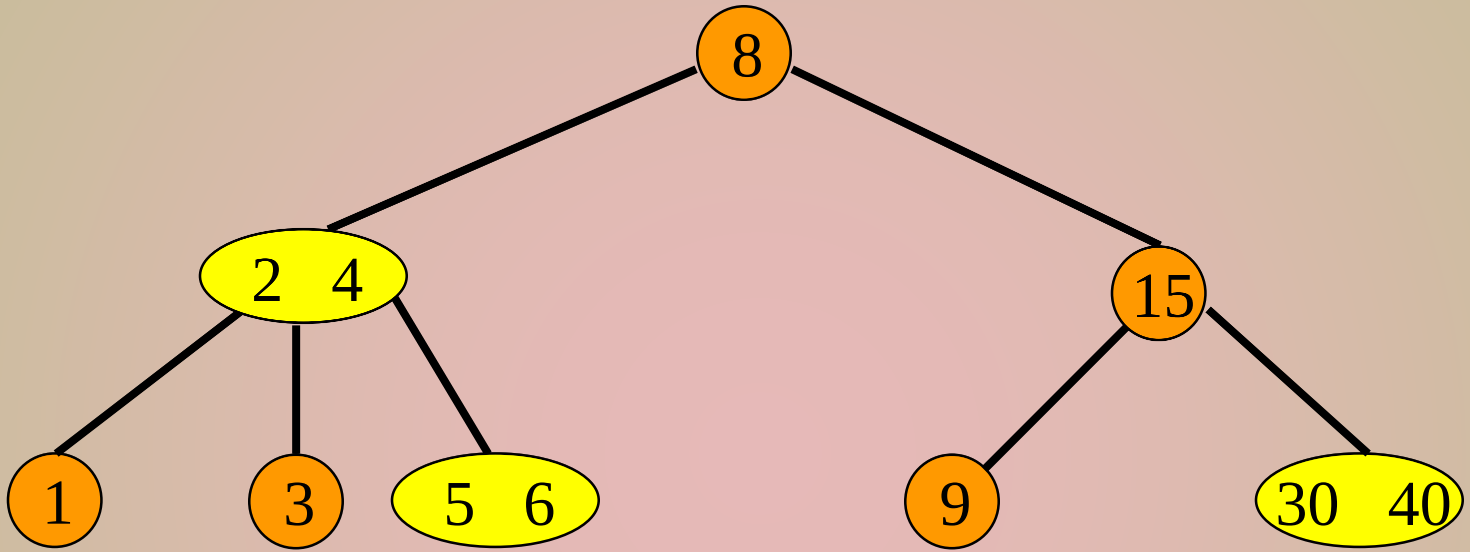
- Delete the pair with key = 17.
- Deletion from a 2-node.
- Check one sibling and determine if it is a 3-node.
- If so borrow a pair and a subtree via parent node.

Delete From A Leaf



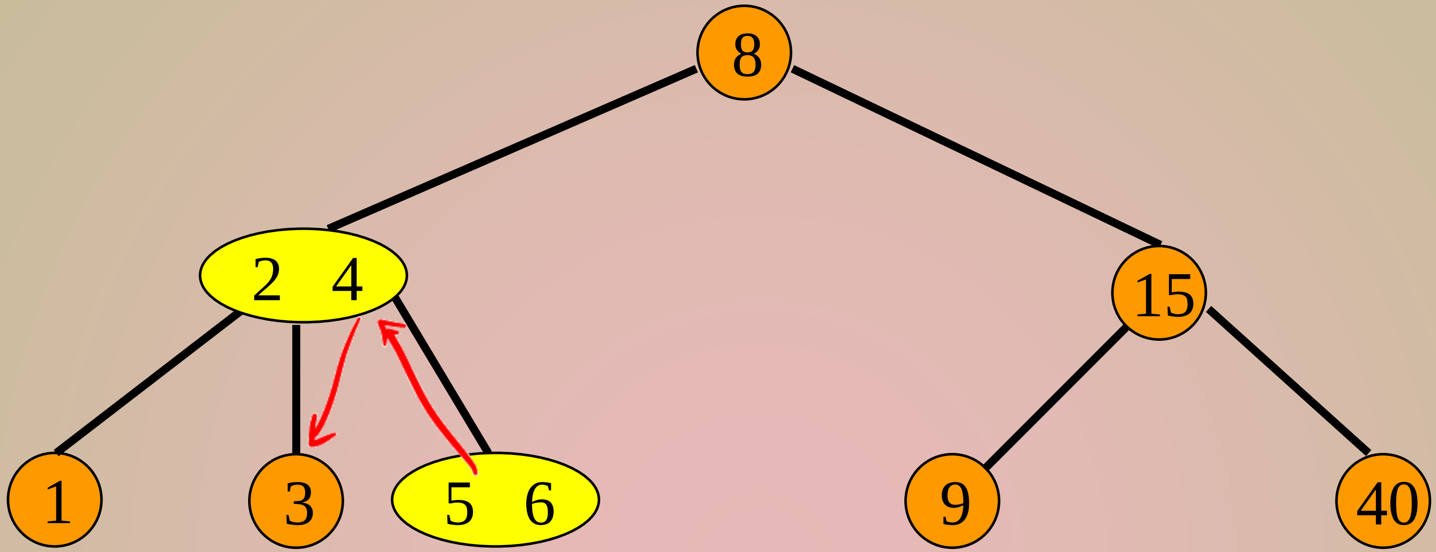
- Delete the pair with key = 20.
- Deletion from a 2-node.
- Check one sibling and determine if it is a 3-node.
- If not, combine with sibling and parent pair.

Delete From A Leaf



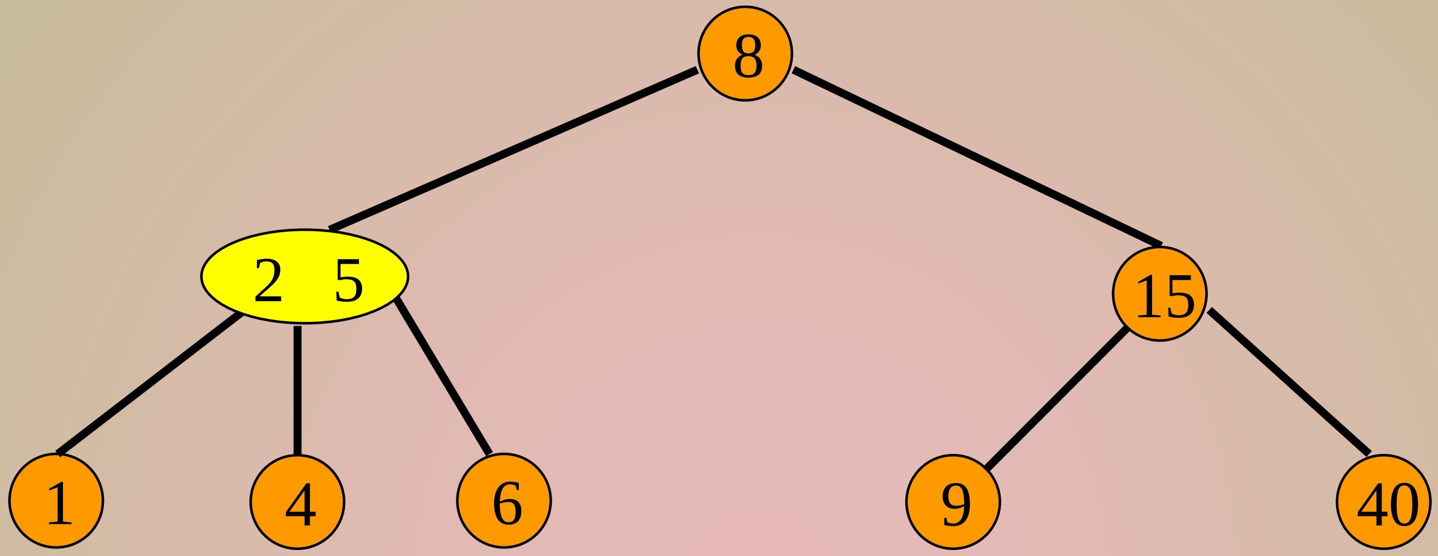
- Delete the pair with key = 30.
- Deletion from a 3-node.
- 3-node becomes 2-node.

Delete From A Leaf



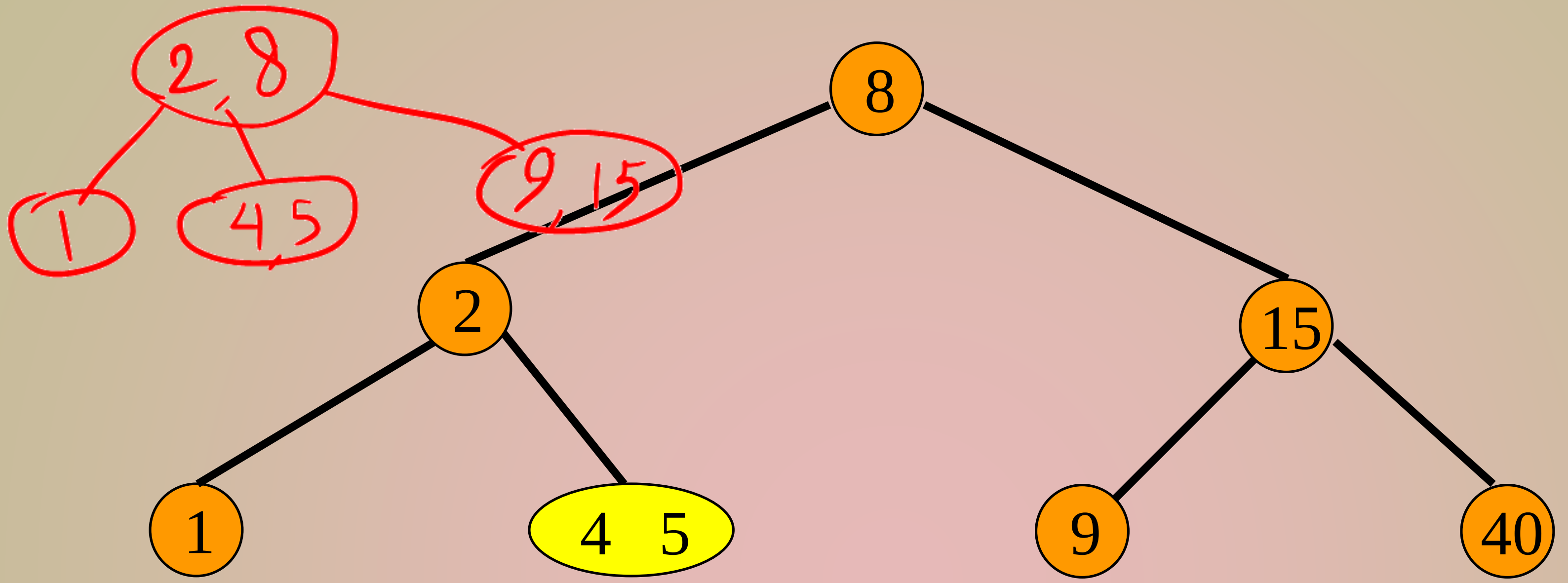
- Delete the pair with key = 3.
- Deletion from a 2-node.
- Check one sibling and determine if it is a 3-node.
- If so borrow a pair and a subtree via parent node.

Delete From A Leaf



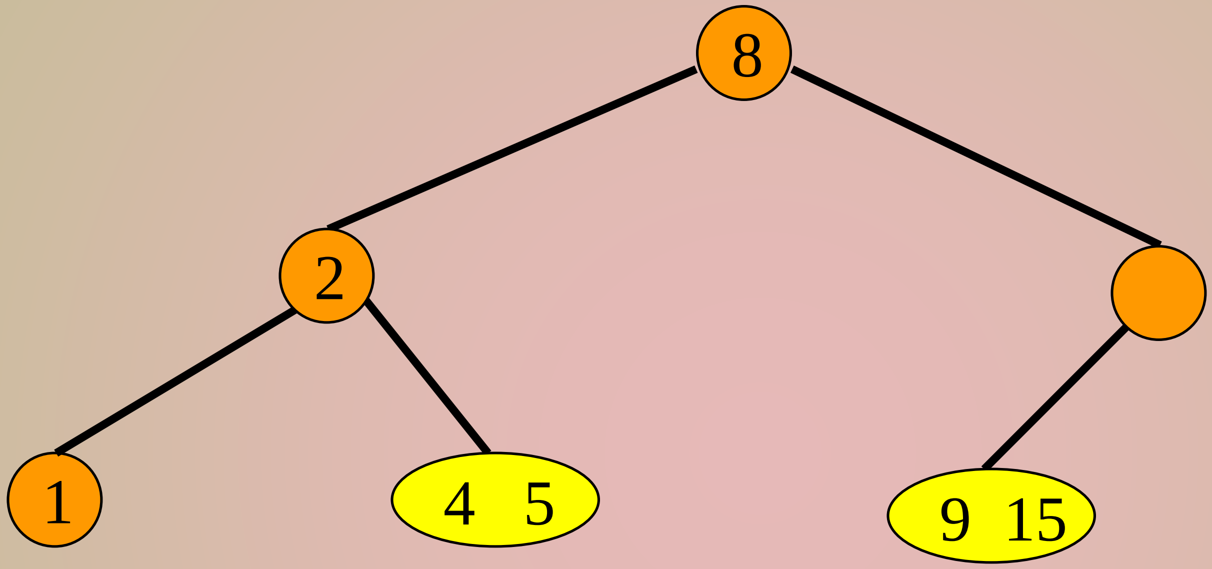
- Delete the pair with key = 6.
- Deletion from a 2-node.
- Check one sibling and determine if it is a 3-node.
- If not, combine with sibling and parent pair.

Delete From A Leaf



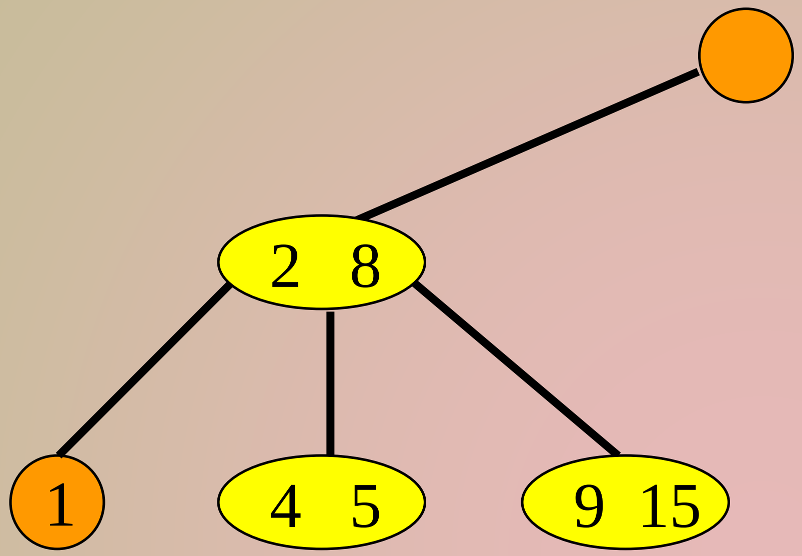
- Delete the pair with key = 40.
- Deletion from a 2-node.
- Check one sibling and determine if it is a 3-node.
- If not, combine with sibling and parent pair.

Delete From A Leaf



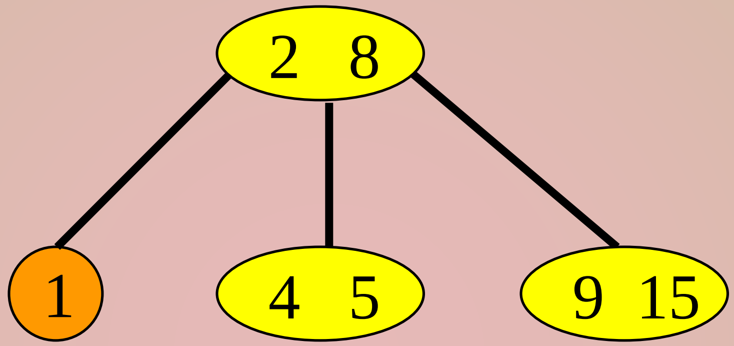
- Parent pair was from a 2-node.
- Check one sibling and determine if it is a 3-node.
- If not, combine with sibling and parent pair.

Delete From A Leaf



- Parent pair was from a 2-node.
- Check one sibling and determine if it is a 3-node.
- No sibling, so must be the root.
- Discard root. Left child becomes new root.

Delete From A Leaf



- Height reduces by 1.