Data Structures and Algorithms
(m-way search tree)

Dr. Sambit Bakshi

Dept. of CSE, NIT Rourkela

m-Way Search Tree

- An **m**-way search tree **T** may be an empty tree.
- If **T** is non-empty, it satisfies the following properties:

(i) For some integer **m** known as the order of the tree, each node has at most **m** child nodes.

(ii) A node may be represented as

$$A_0 , (K_1, A_1), (K_2, A_2) \ldots . (K_{m-1} , A_{m-1})$$

where $K_i , 1 \leq i \leq m-1$ are the keys and A_i, $0 \leq i \leq m-1$ are the pointers to the subtree of **T**

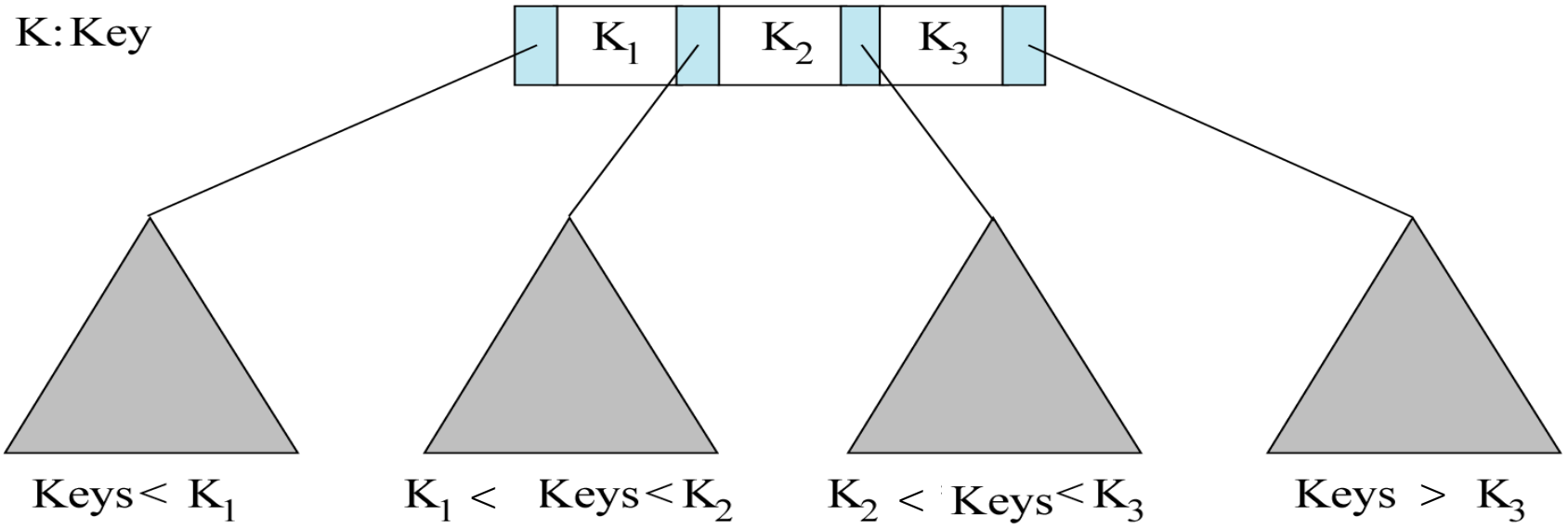
(iii) If the node has **c** child nodes where $c \leq m$, then the node can have only (**c-1**) keys, $K_1 , K_2 , \ldots\ldots K_{c-1}$

(iv) The keys in a node are ordered, i.e., $K_1 < K_2 < \ldots\ldots < K_{c-1}$

m-Way Search Tree

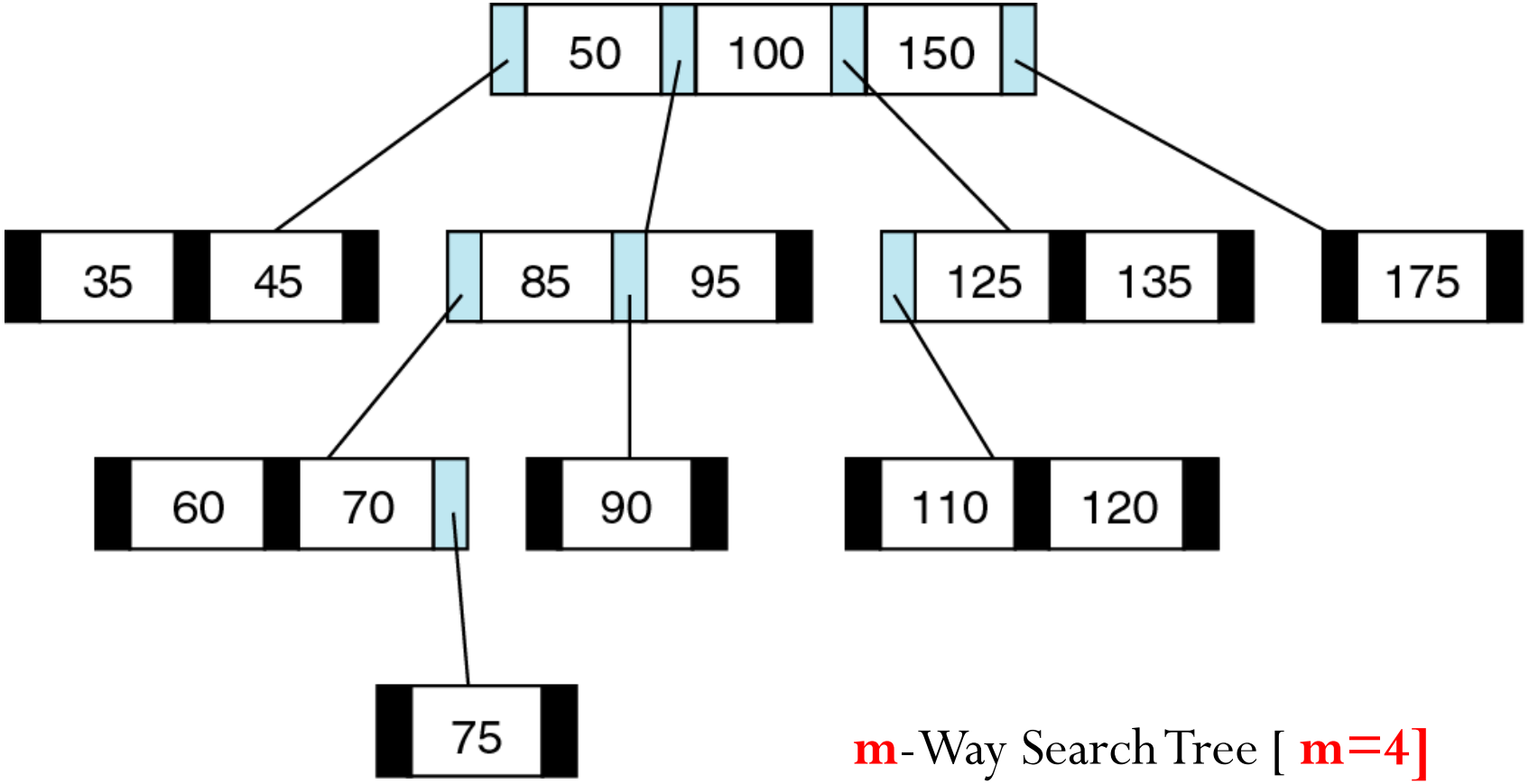
(v) For a node A_0 , $(K_1$, $A_1)$, $(K_2$, $A_2)$, …. $(K_{m-1}$, A_{m-1}), if S_i is the subtree pointed by A_i, $0 \leq i \leq m-1$then

- $Key(S_0) < K_1$
- $Key(S_{m-1}) > K_{m-1}$
- $K_i < Key(S_i) < K_{i+1}$, $1 \leq i \leq m-2$



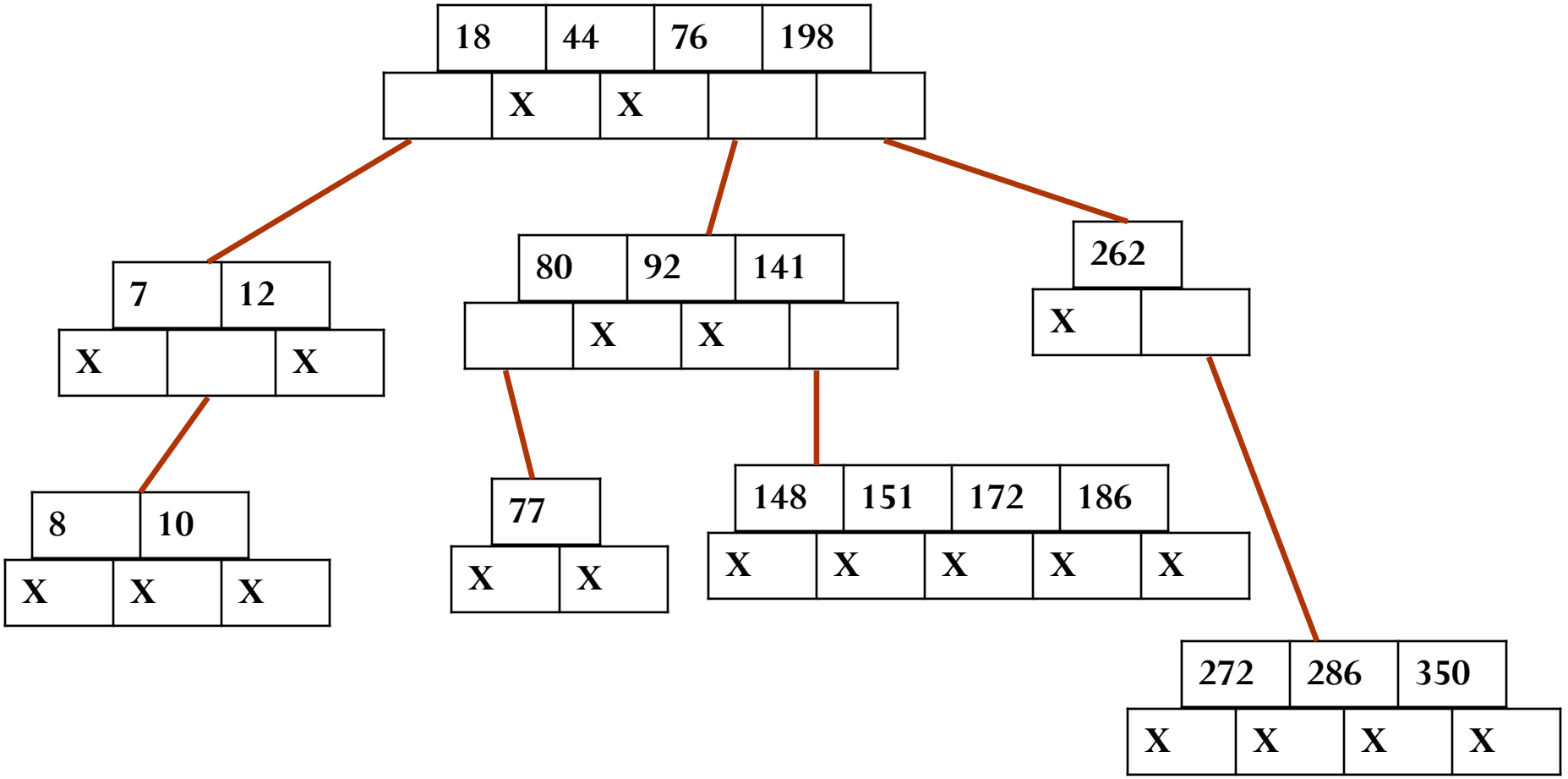
m-Way Search Tree

(vi) Each of the subtree A_i , $0 \leq i \leq m-1$ are also m-way search tree



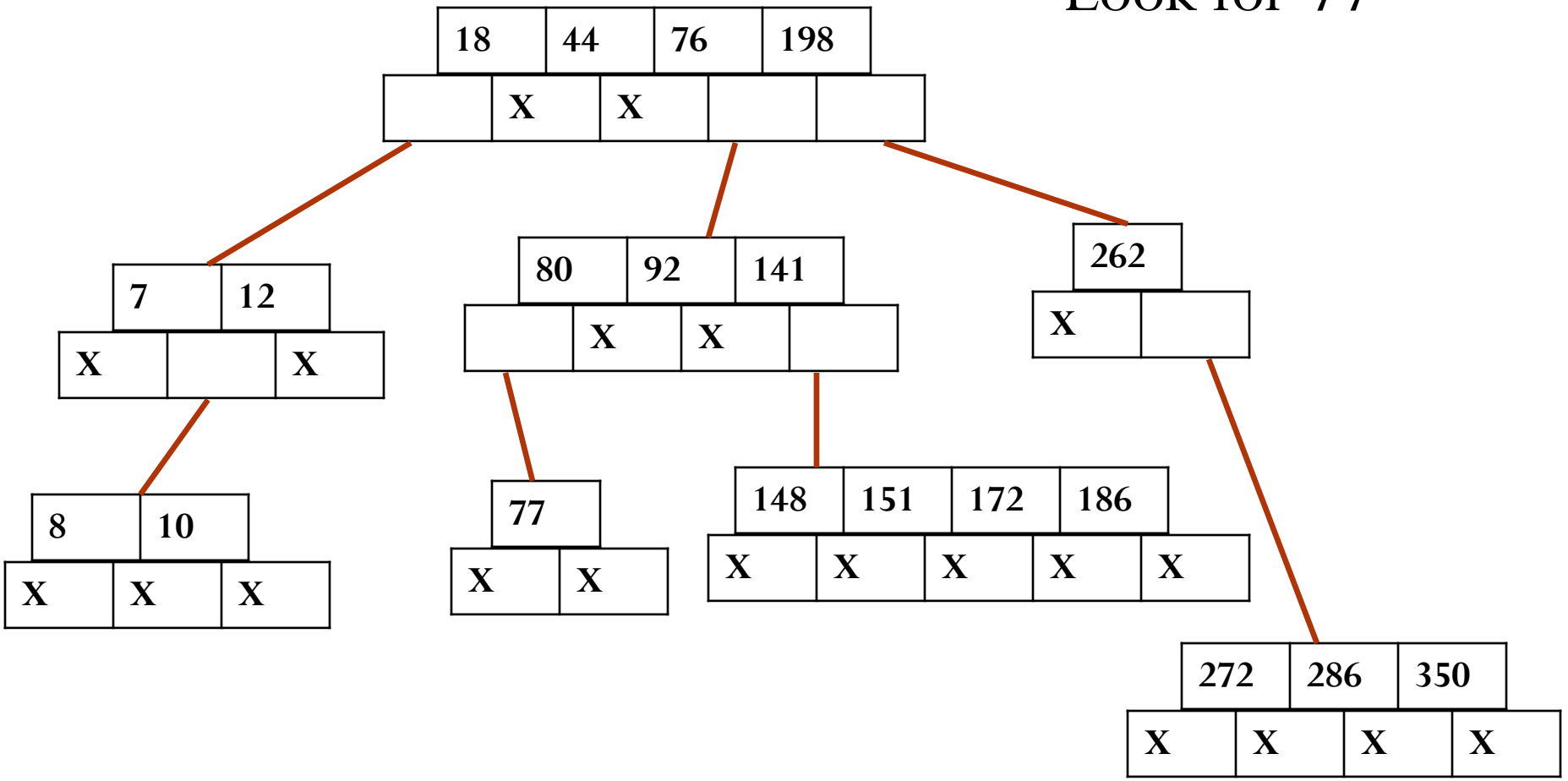
m-Way Search Tree [**m=4**]

m-Way Search Tree [m=5]

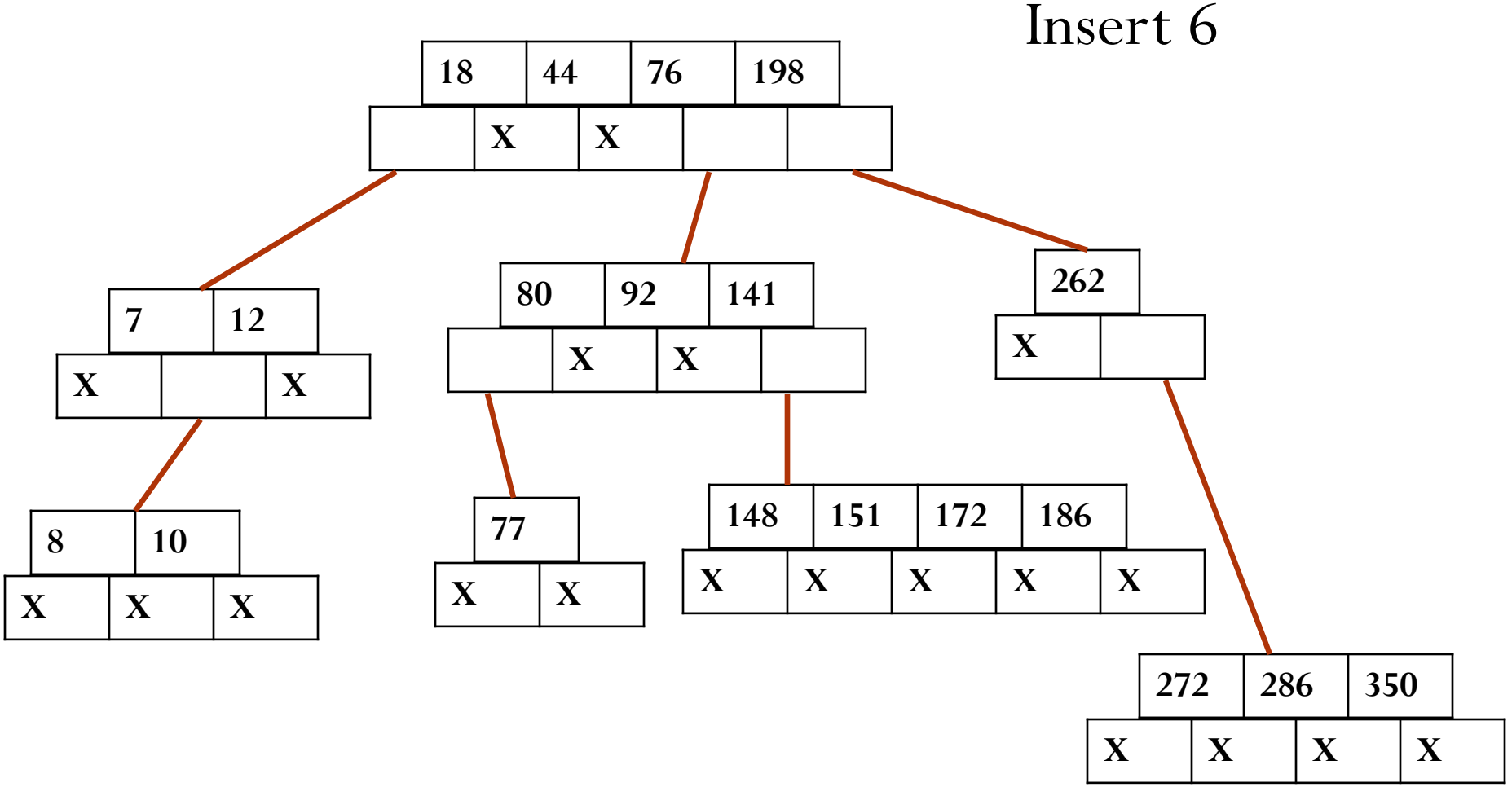


Searching in an m-Way Search Tree

Look for 77



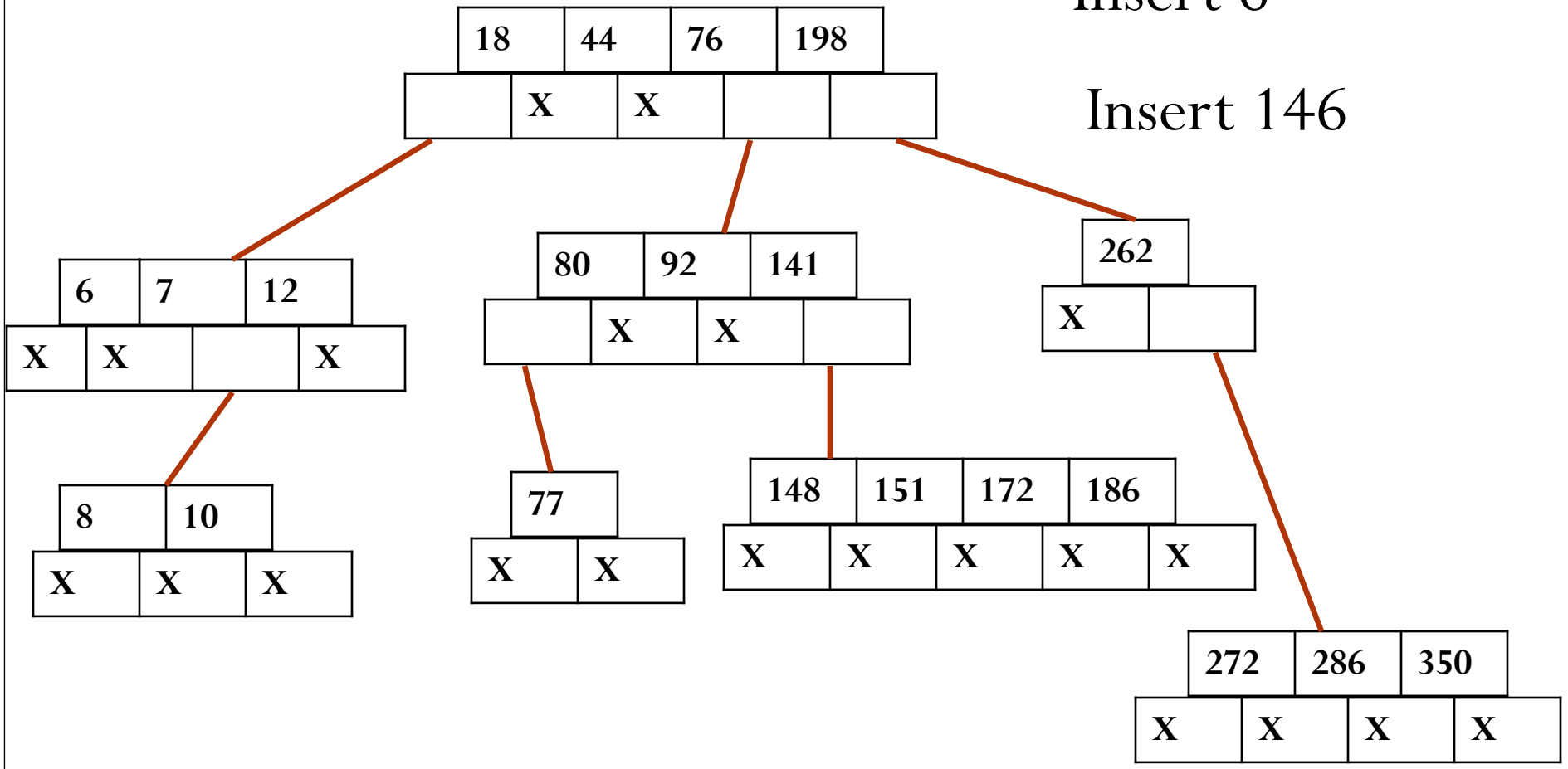
Insertion in an m-Way Search Tree



Insertion in an m-Way Search Tree

Insert 6

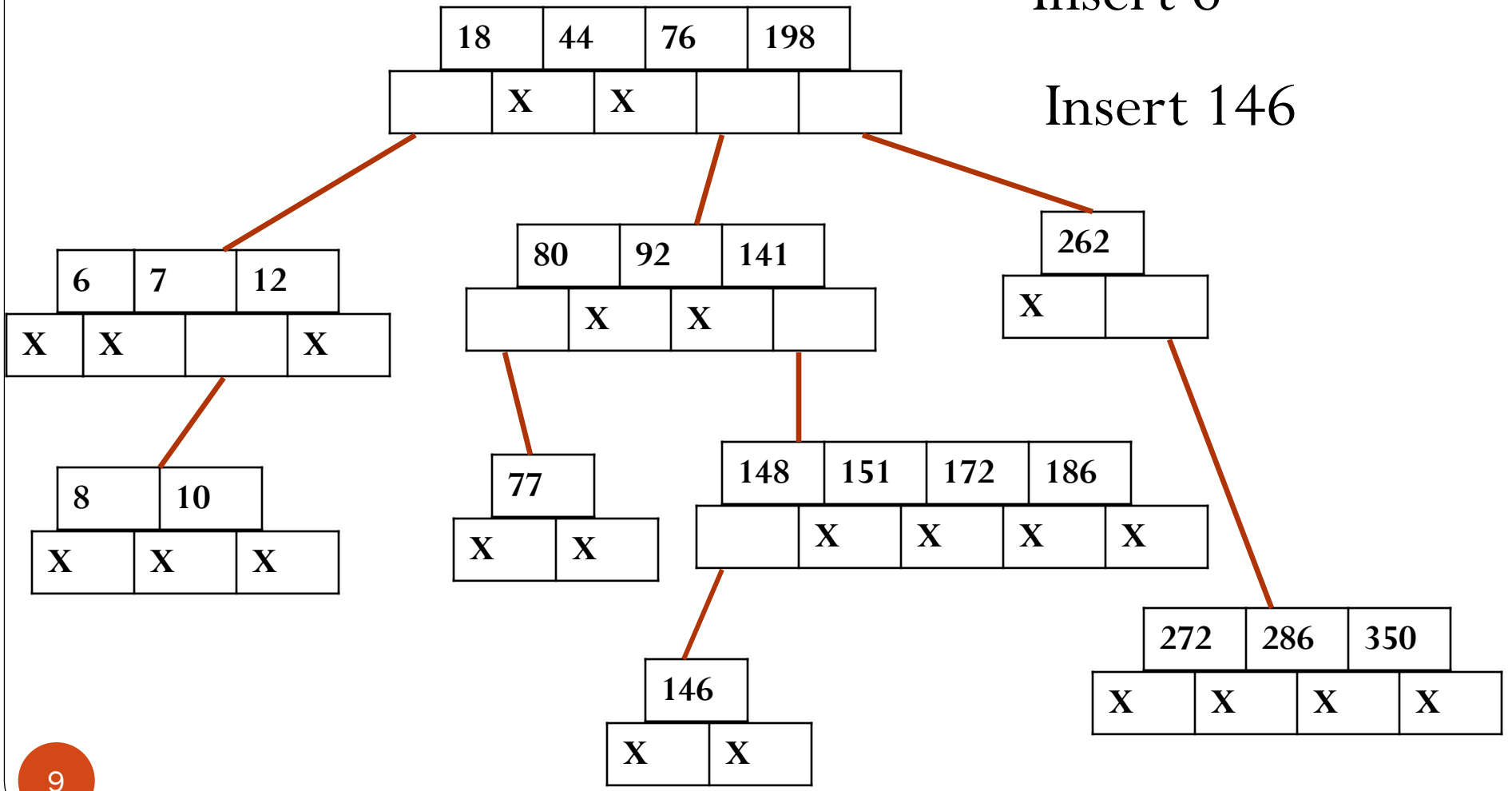
Insert 146



Insertion in an m-Way Search Tree

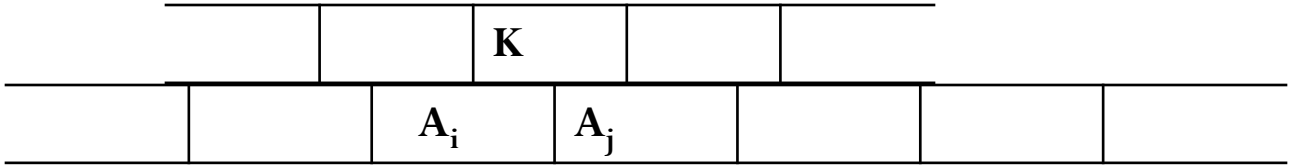
Insert 6

Insert 146



Deletion in an m-Way Search Tree

Let K be the key to be deleted from the m-way search tree.



K : Key

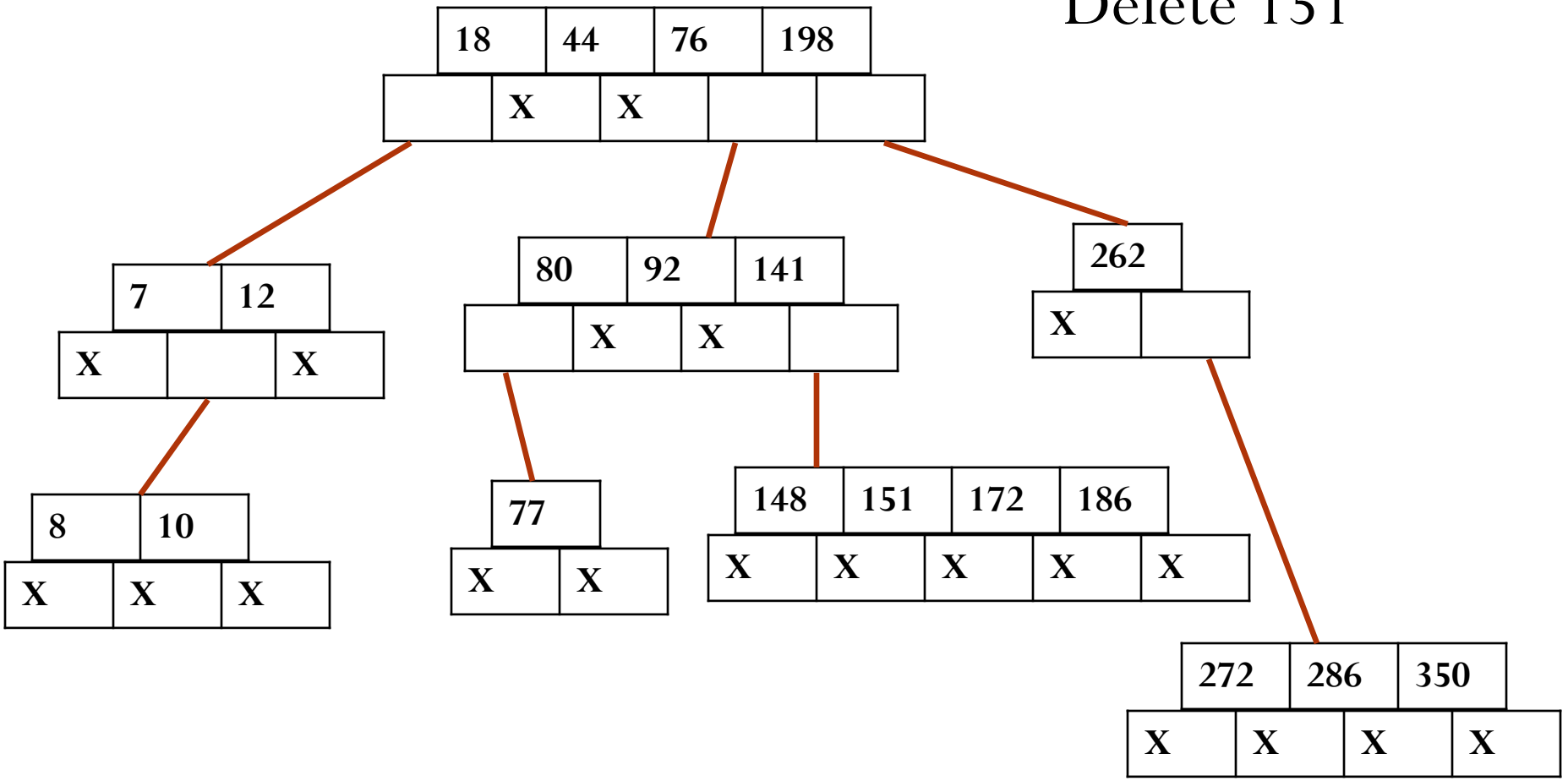
A_i , A_j : Pointers to subtree

Deletion in an m-Way Search Tree

1) If $(A_i = A_j = NULL)$ then delete **K**
2) If $(A_i \neq NULL, A_j = NULL)$ then choose the largest of the key elements K' in the subtree pointed to by A_i and replace **K** by K'.
3) If $(A_i = NULL, A_j \neq NULL)$ then choose the smallest of the key element K'' from the subtree pointed to by A_j , delete K'' and replace **K** by K''.
4) If $(A_i \neq NULL, A_j \neq NULL)$ then choose the largest of the key elements K' in the subtree pointed to by A_i or the smallest of the key element K'' from the subtree pointed to by A_j **to** replace **K**.

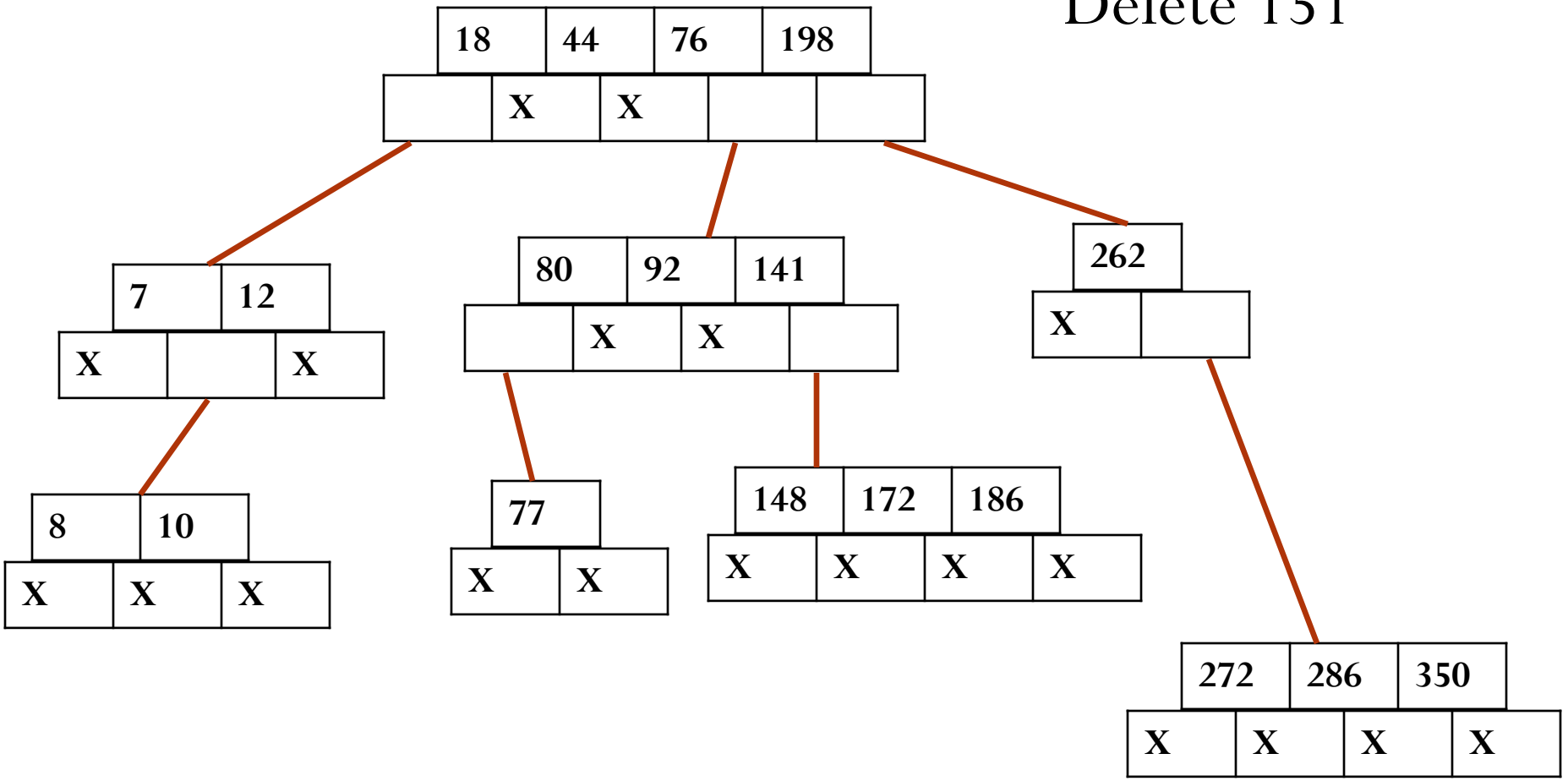
5-Way Search Tree

Delete 151

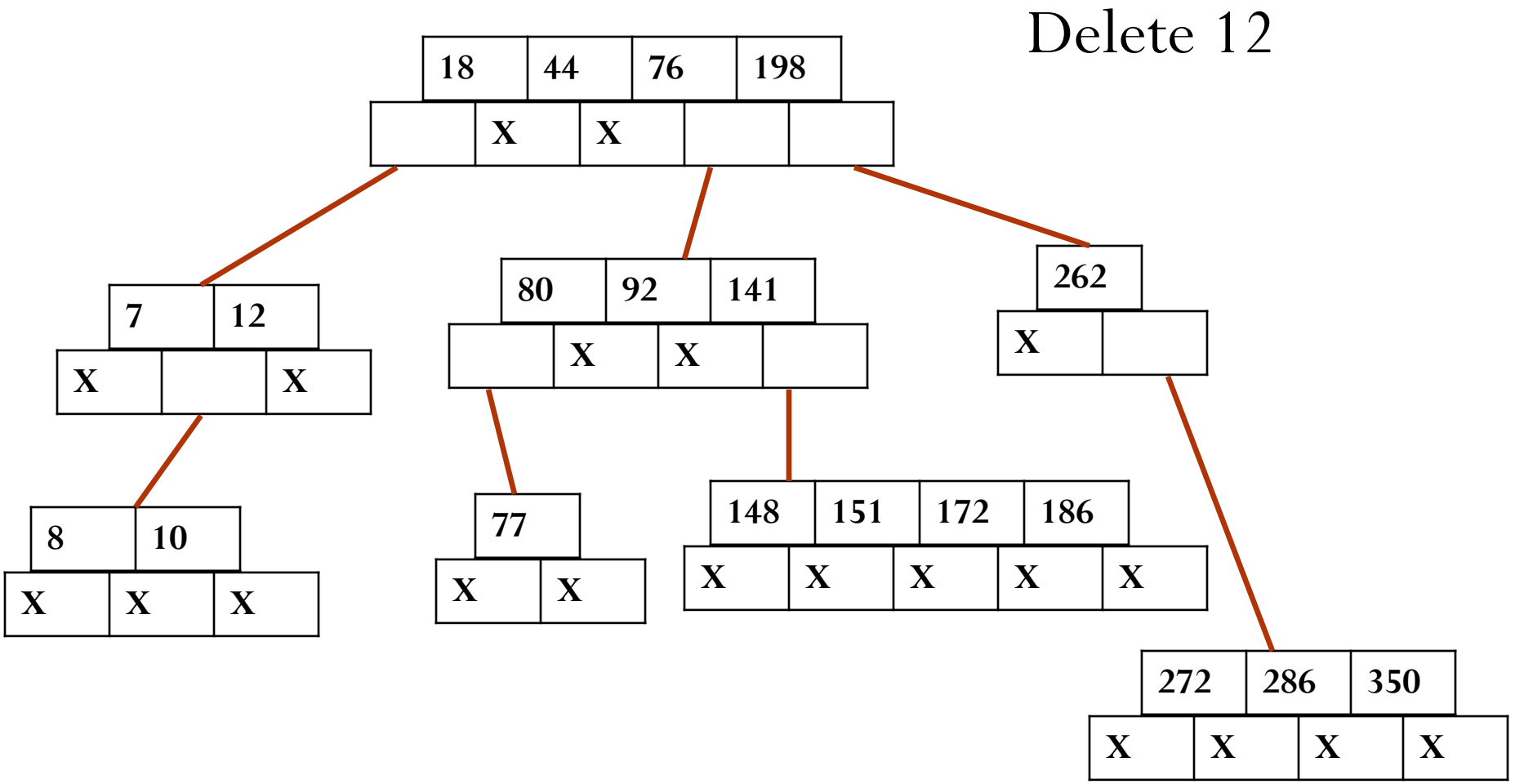


5-Way Search Tree

Delete 151

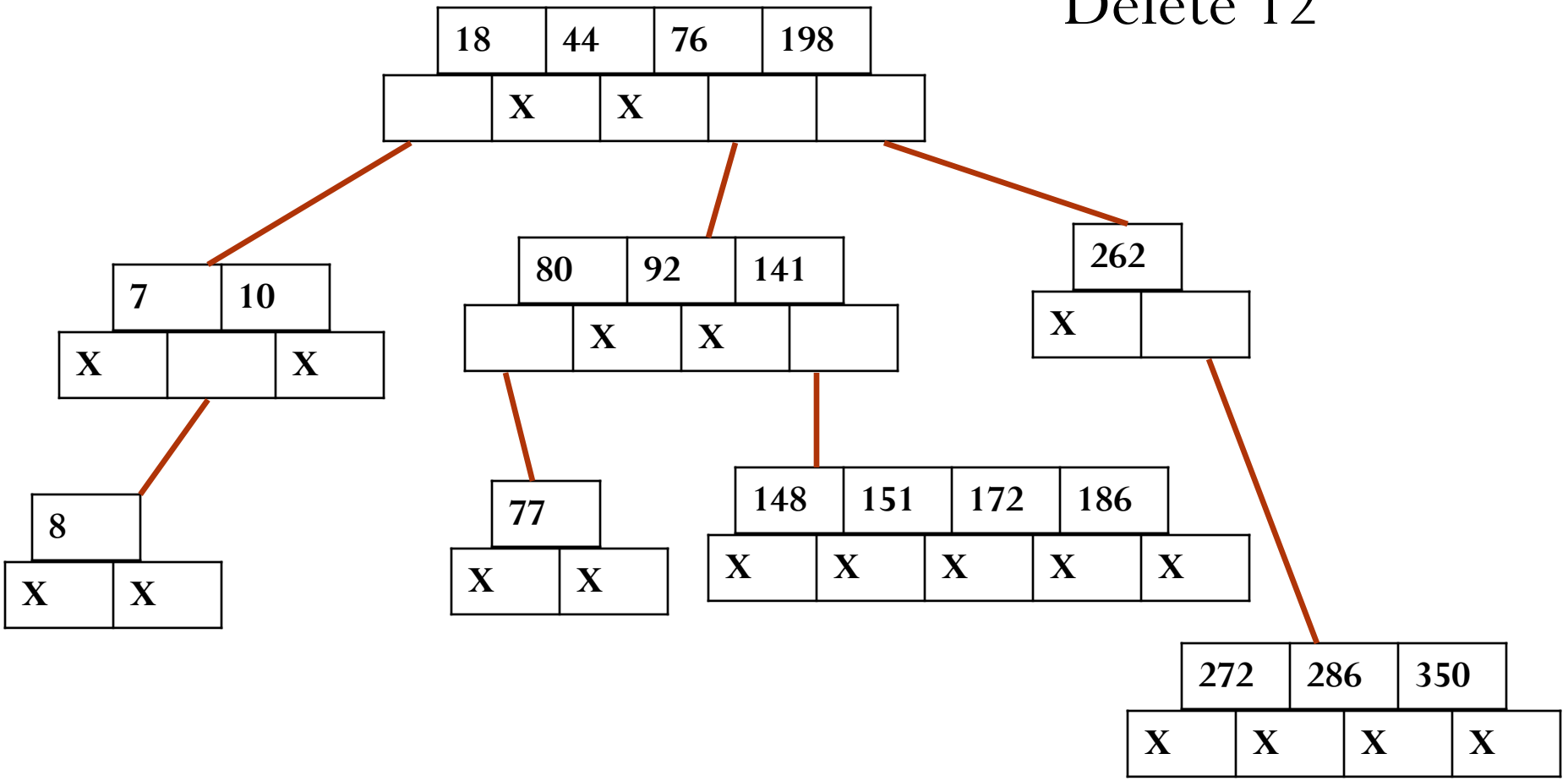


5-Way Search Tree

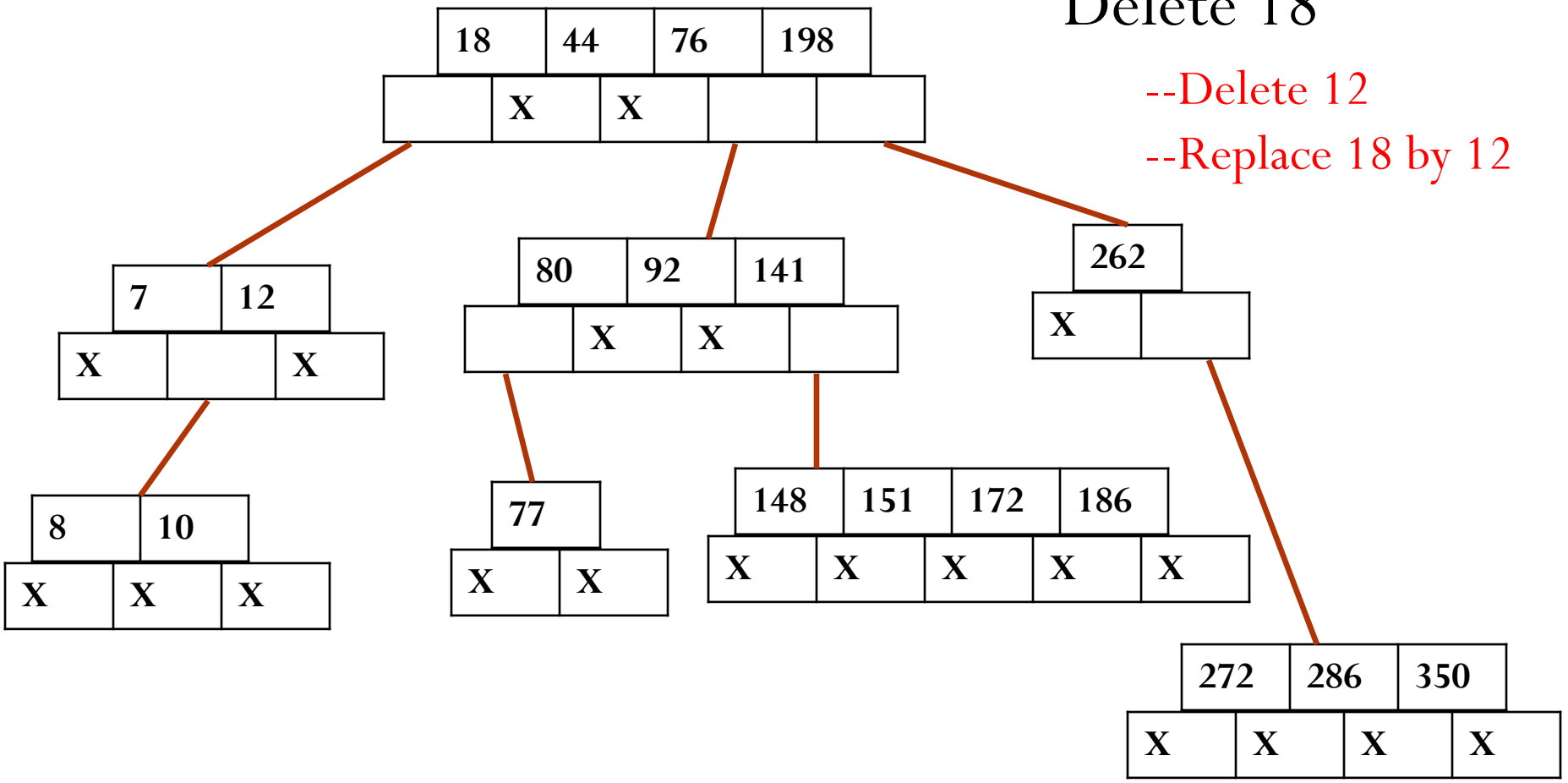


5-Way Search Tree

Delete 12



5-Way Search Tree

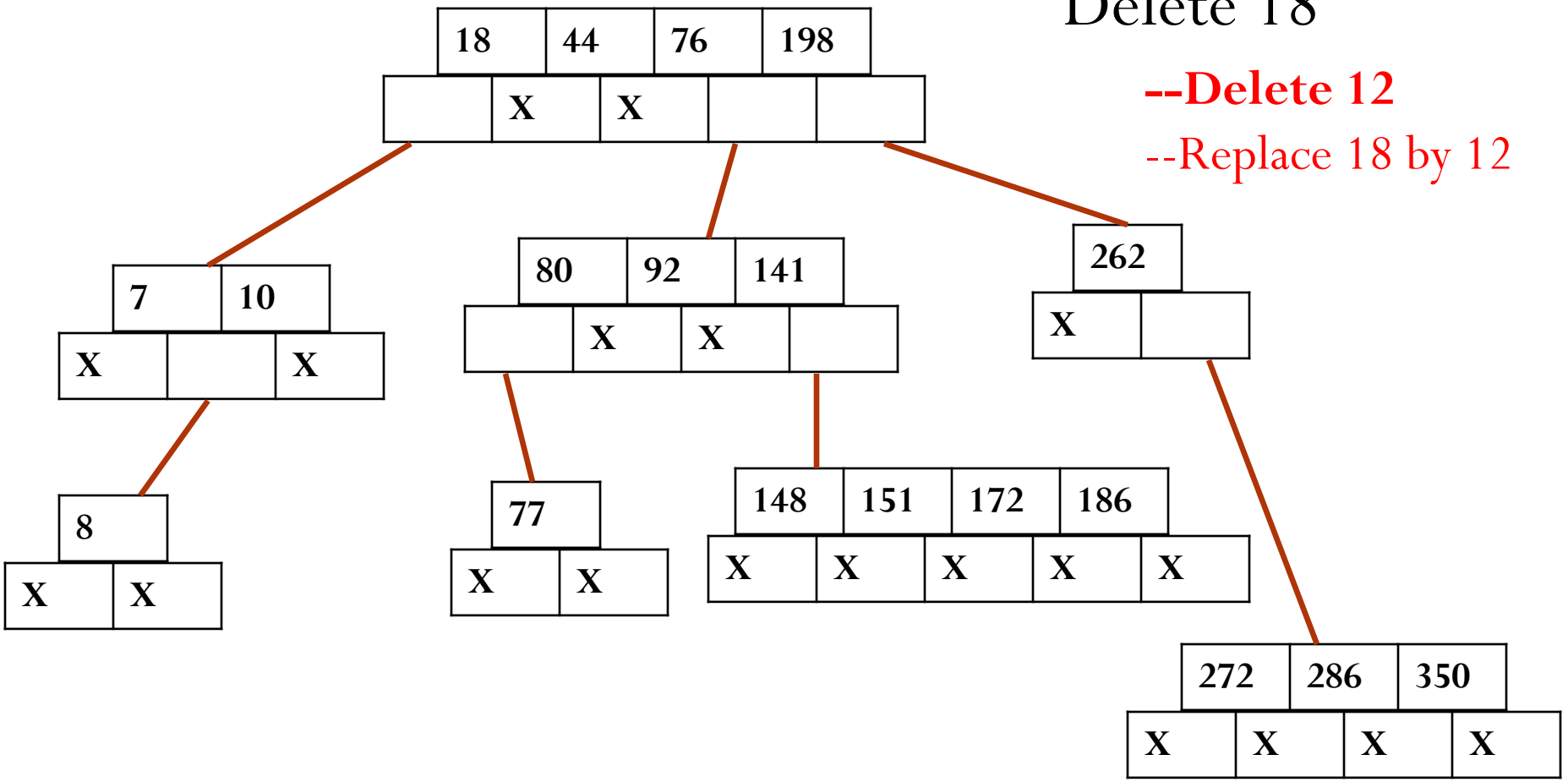


Delete 18

--Delete 12

--Replace 18 by 12

5-Way Search Tree

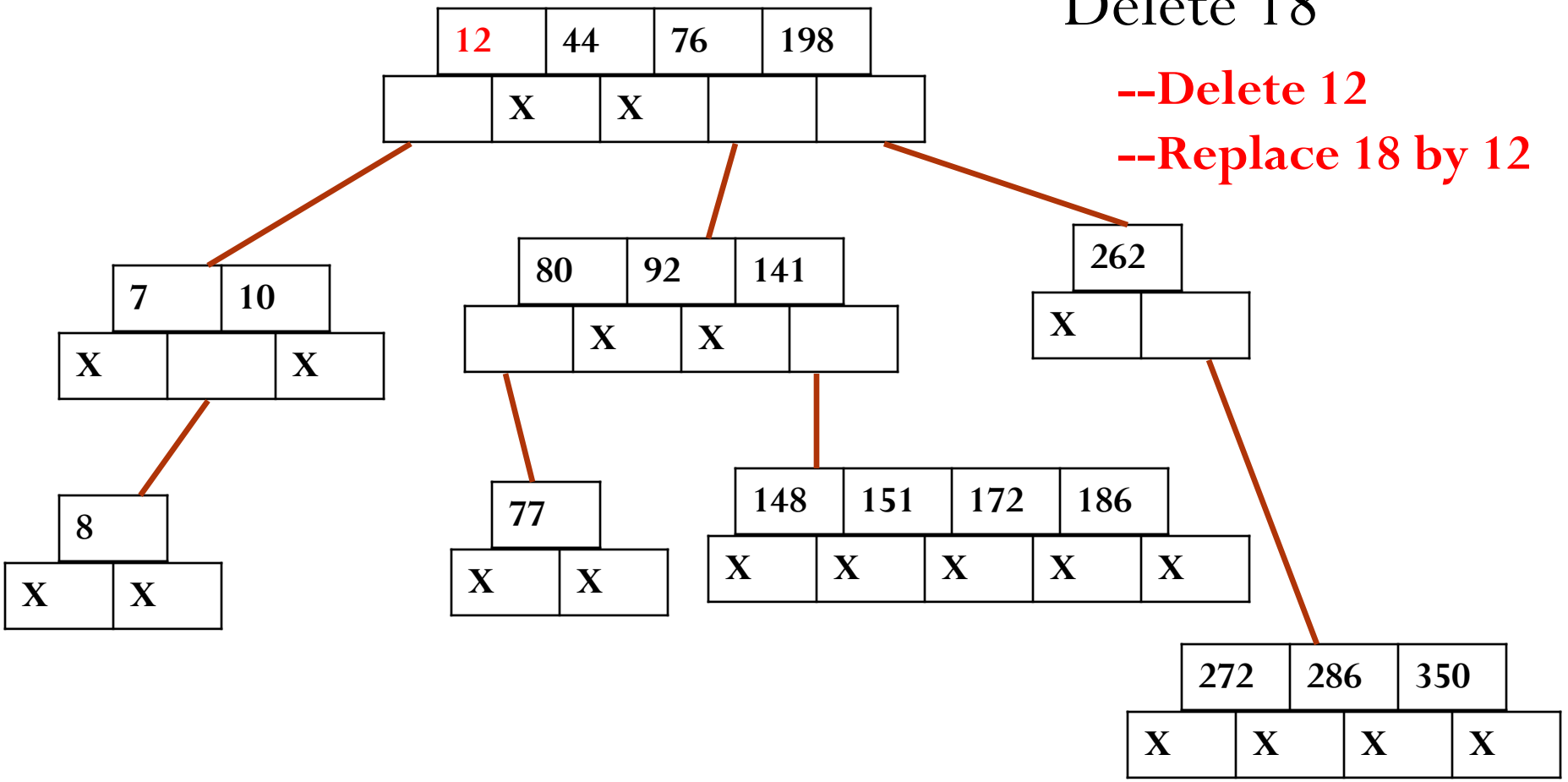


Delete 18

--Delete 12

--Replace 18 by 12

5-Way Search Tree

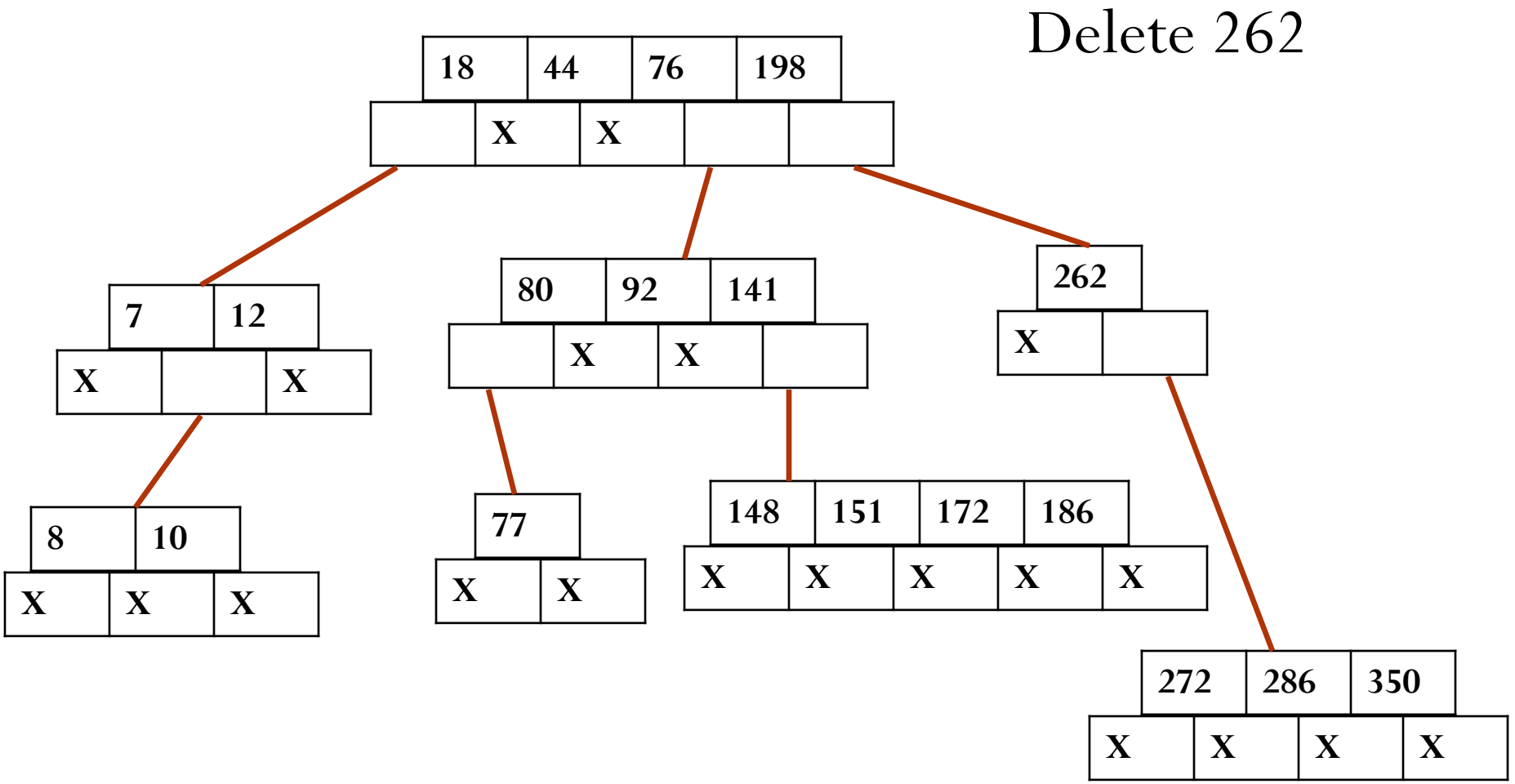


Delete 18

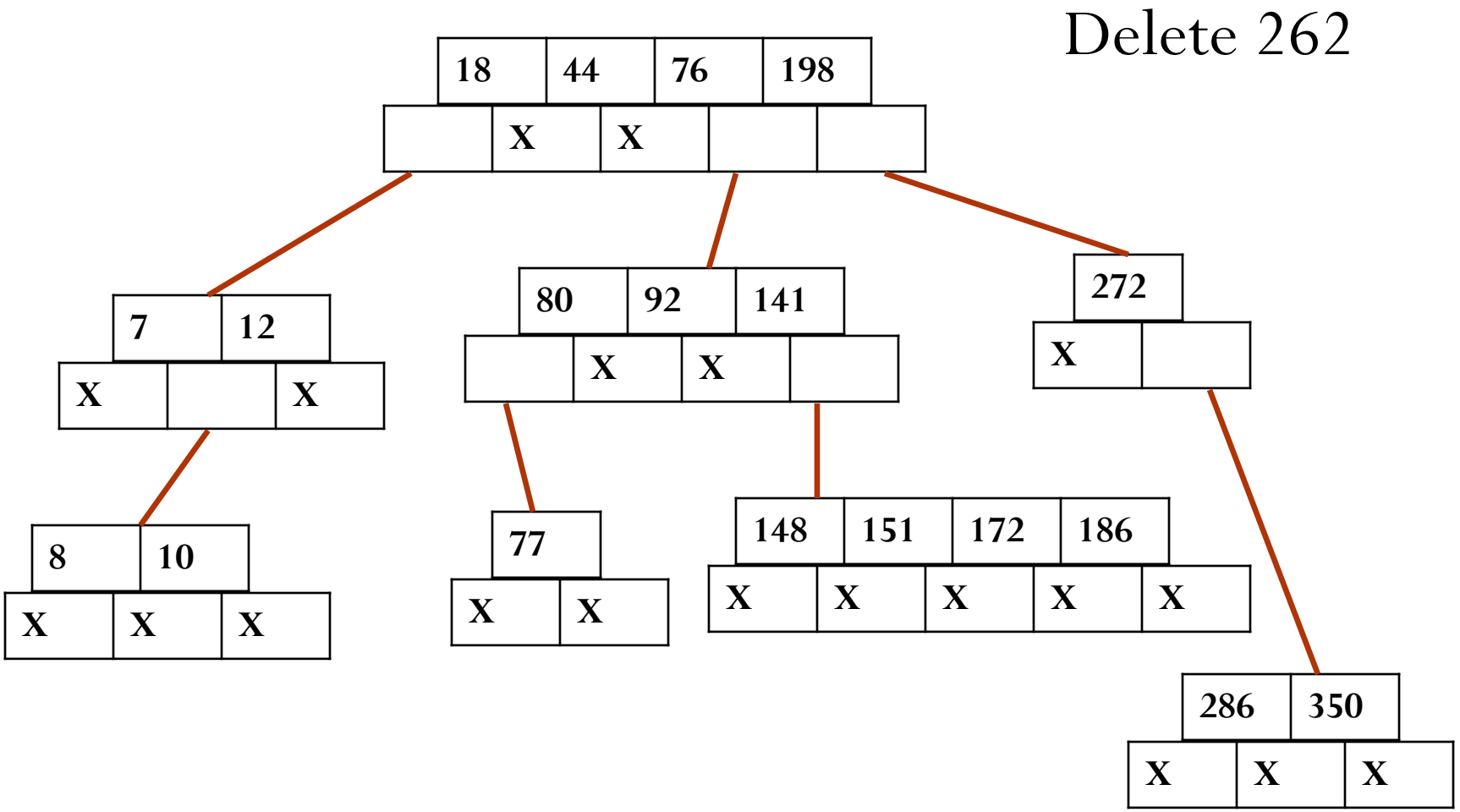
--Delete 12

--Replace 18 by 12

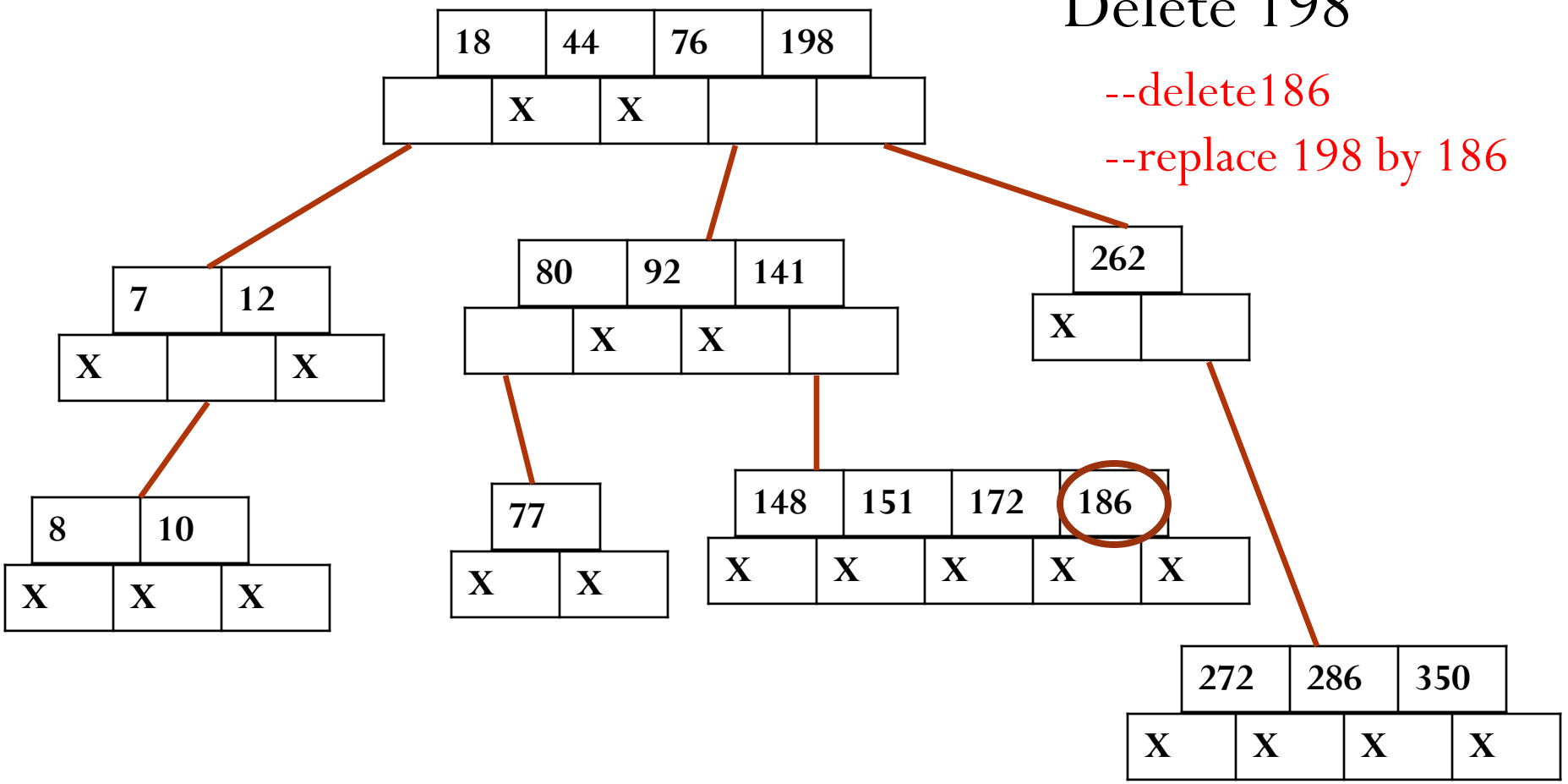
5-Way Search Tree



5-Way Search Tree



5-Way Search Tree

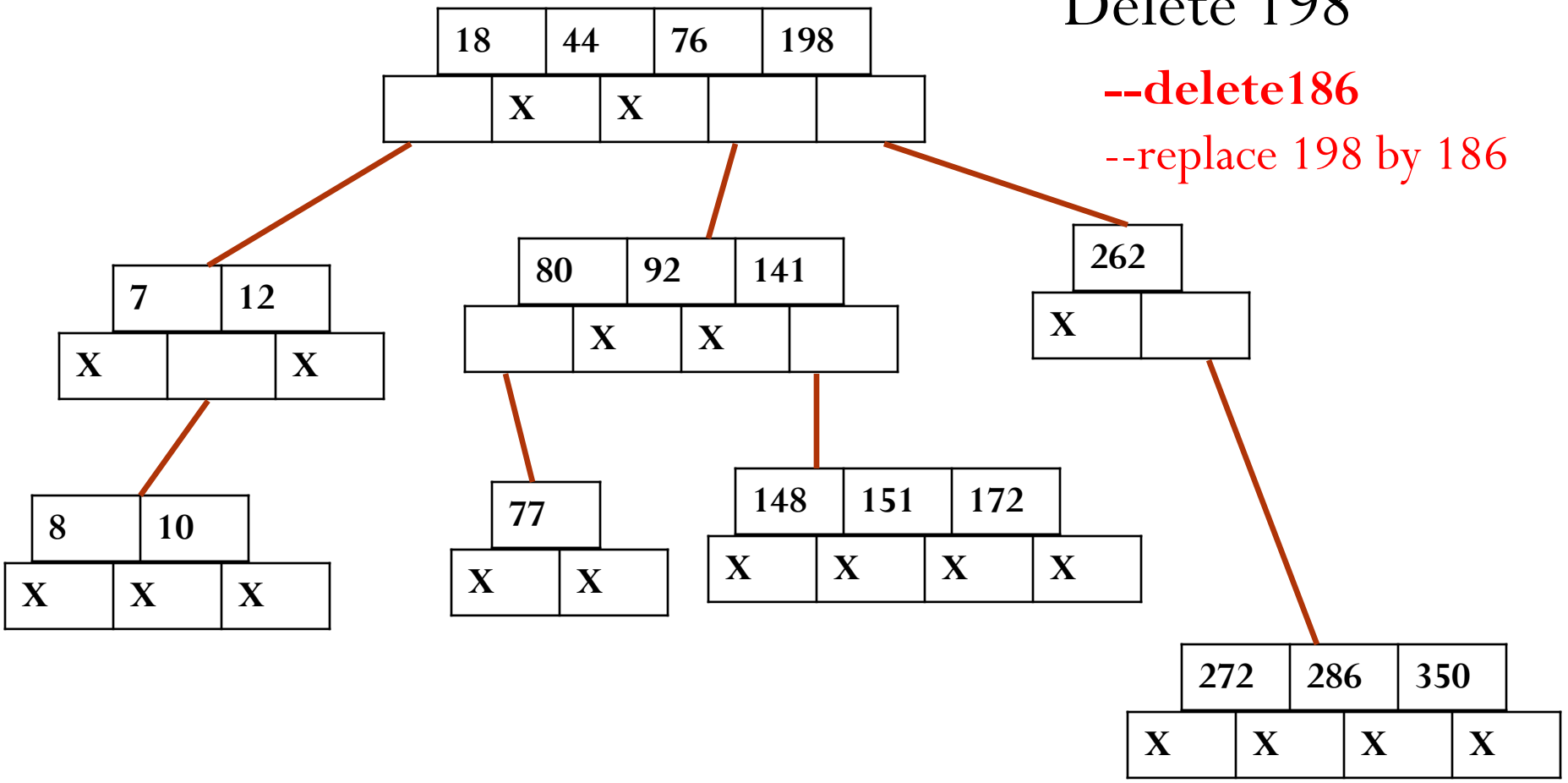


Delete 198

--delete186

--replace 198 by 186

5-Way Search Tree

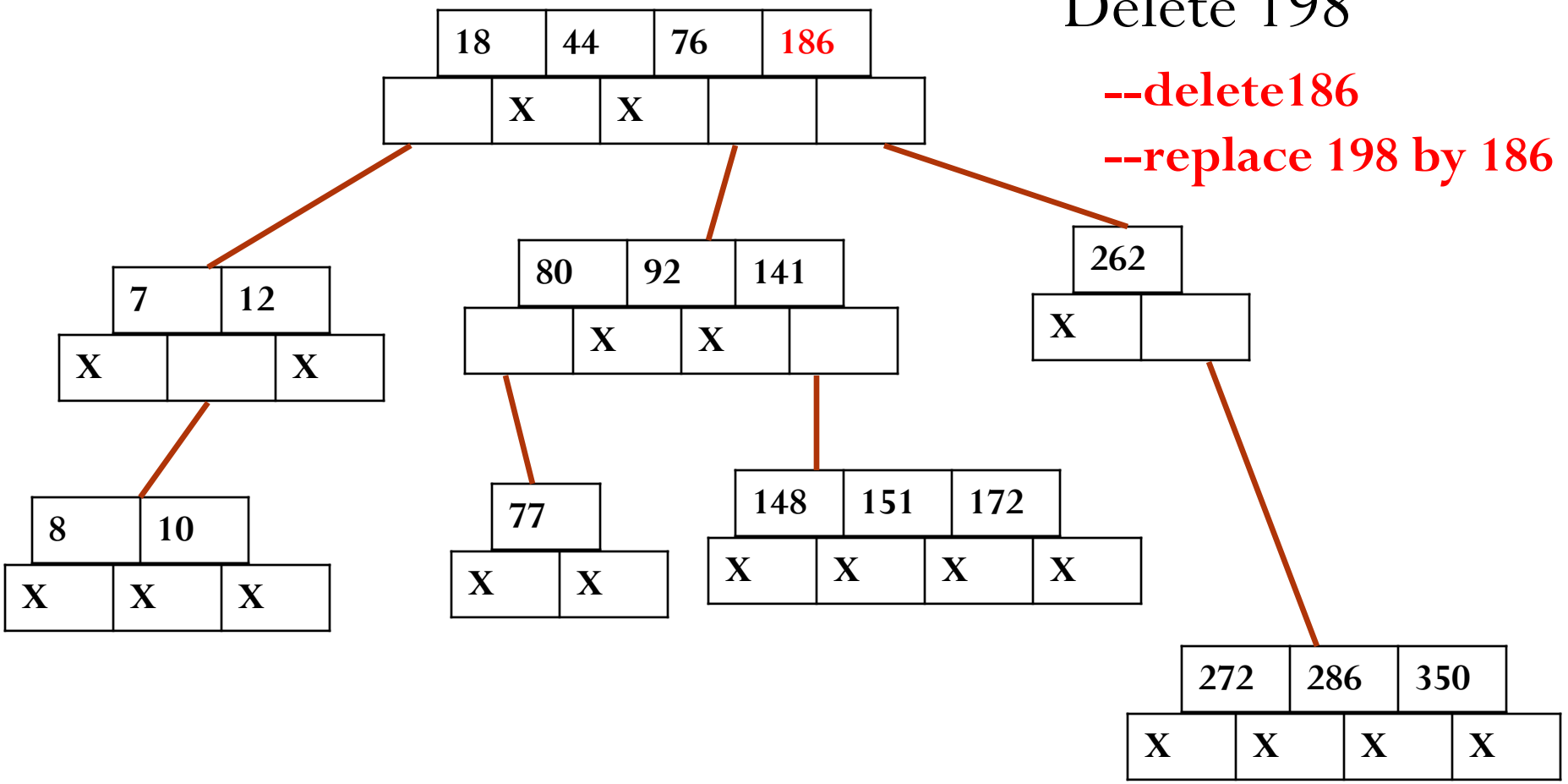


Delete 198

--delete186

--replace 198 by 186

5-Way Search Tree

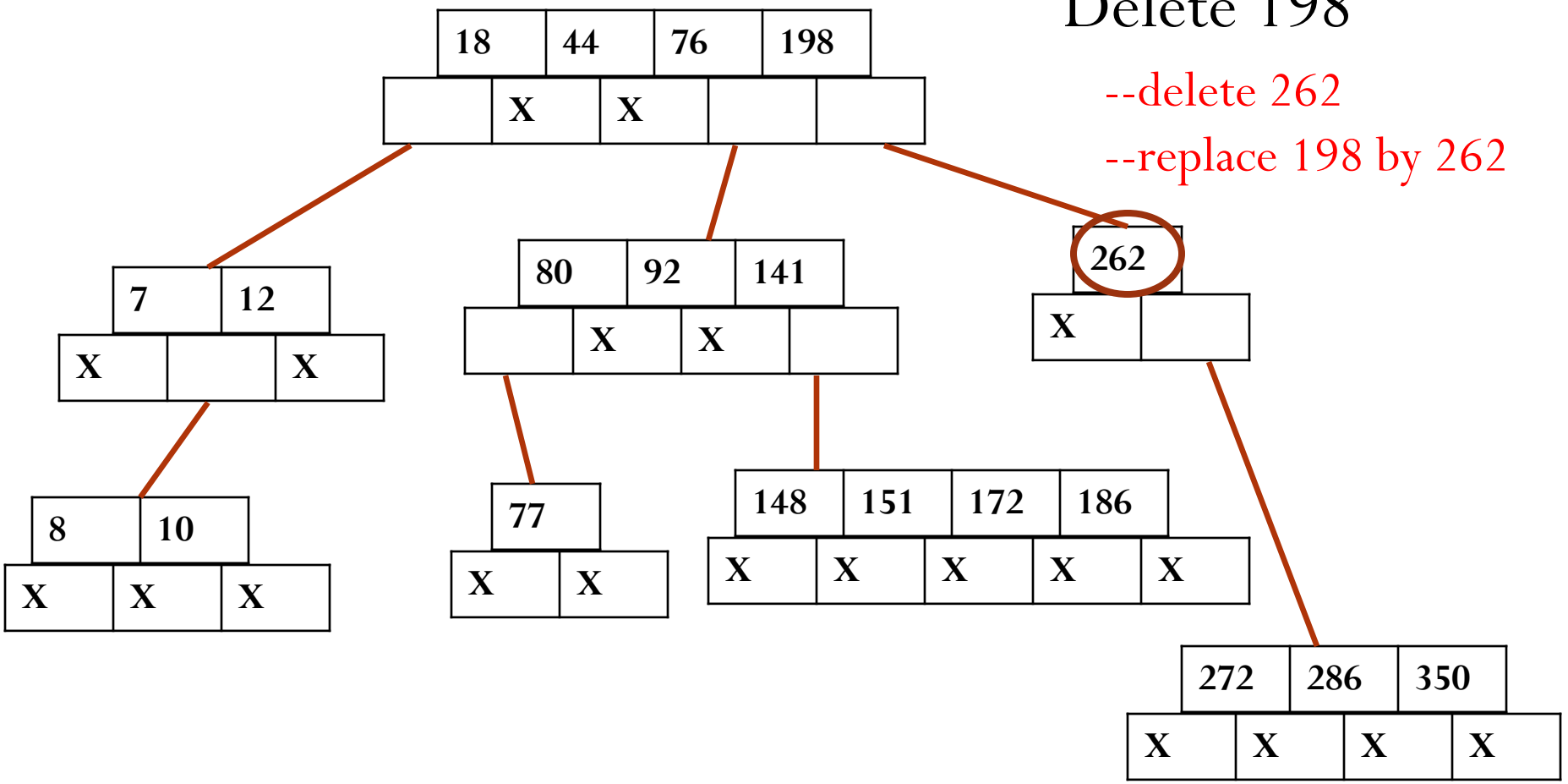


Delete 198

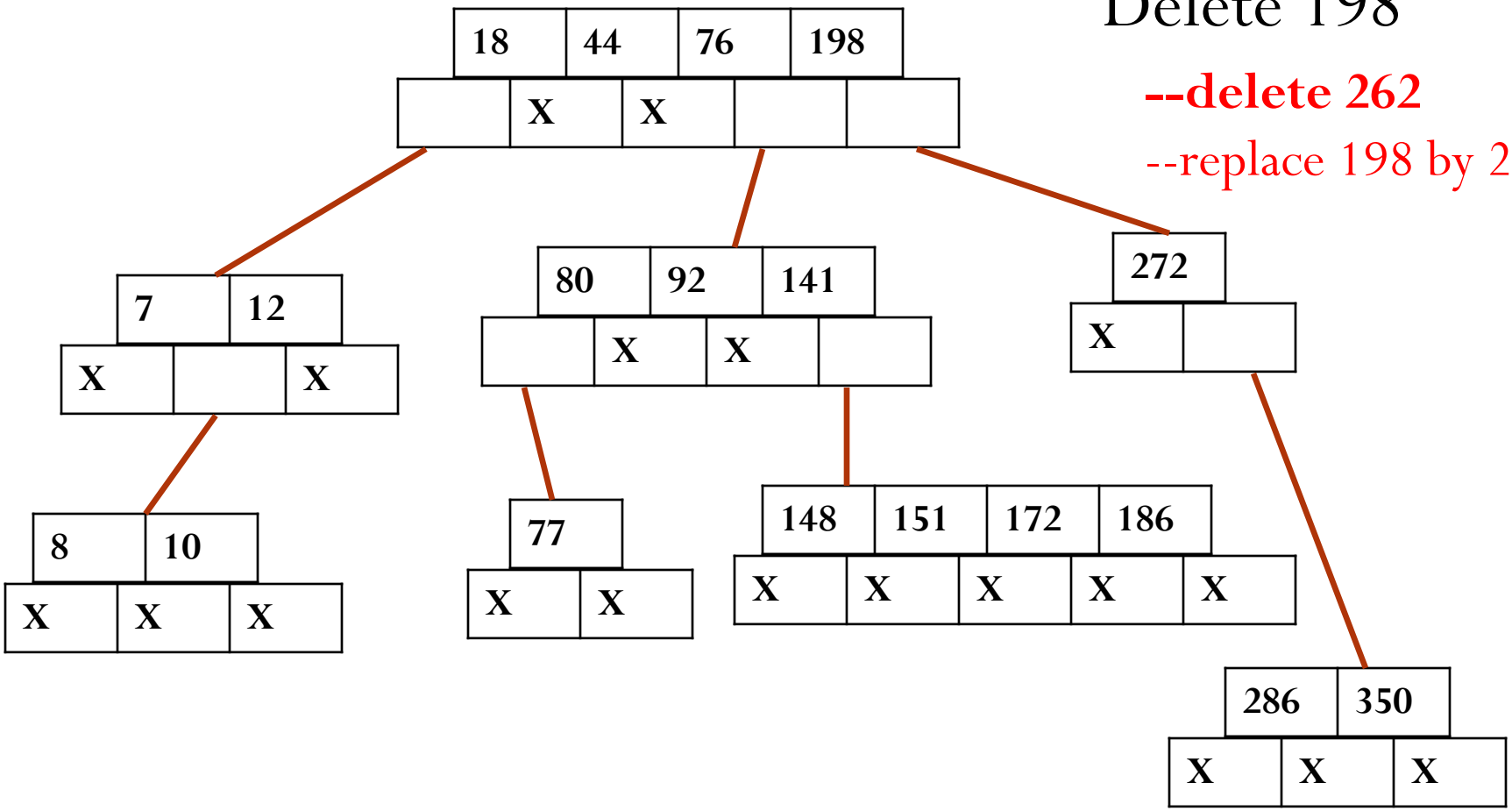
--delete186

--replace 198 by 186

5-Way Search Tree



5-Way Search Tree

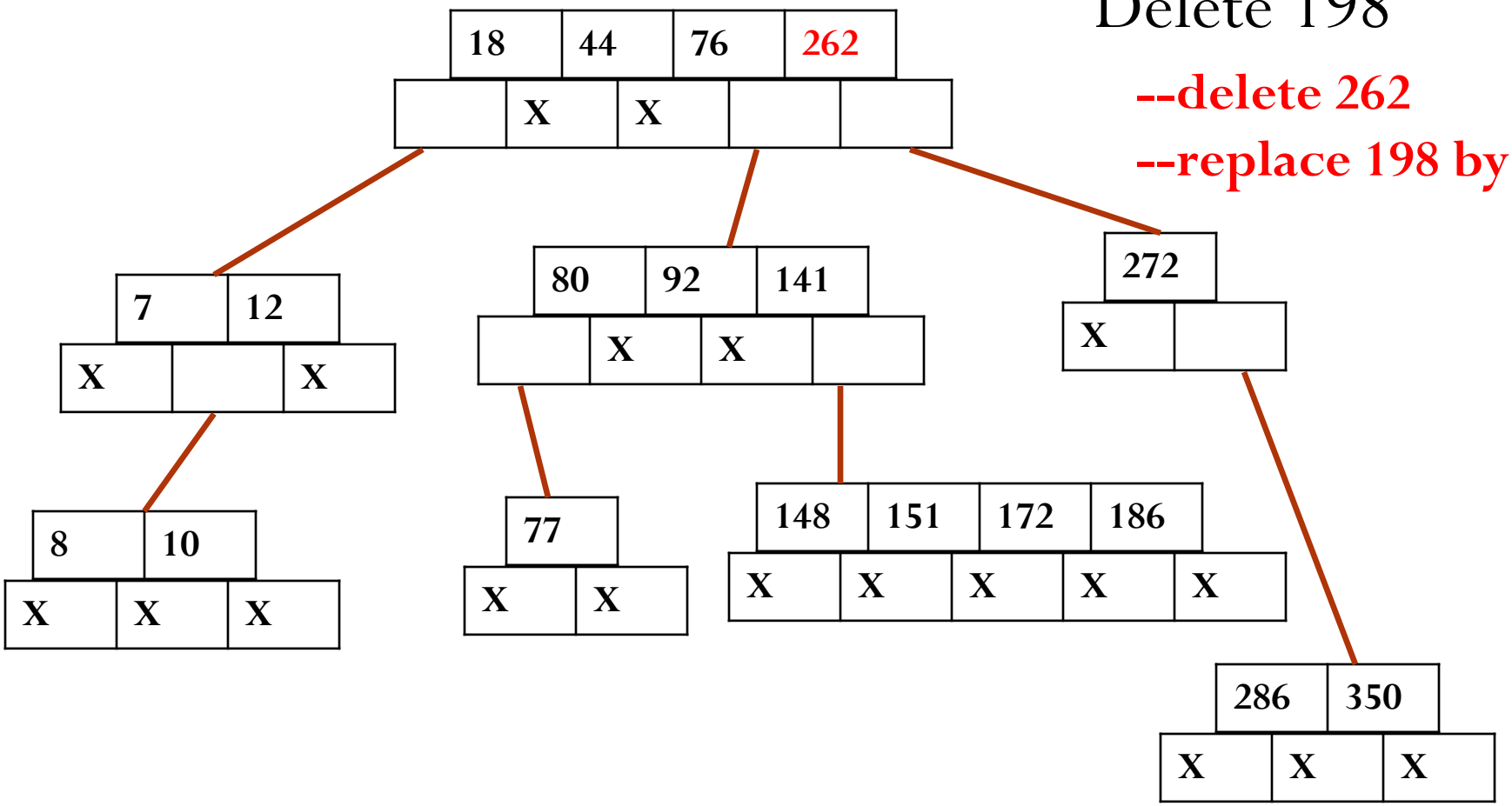


Delete 198

--delete 262

--replace 198 by 262

5-Way Search Tree



Delete 198

--delete 262

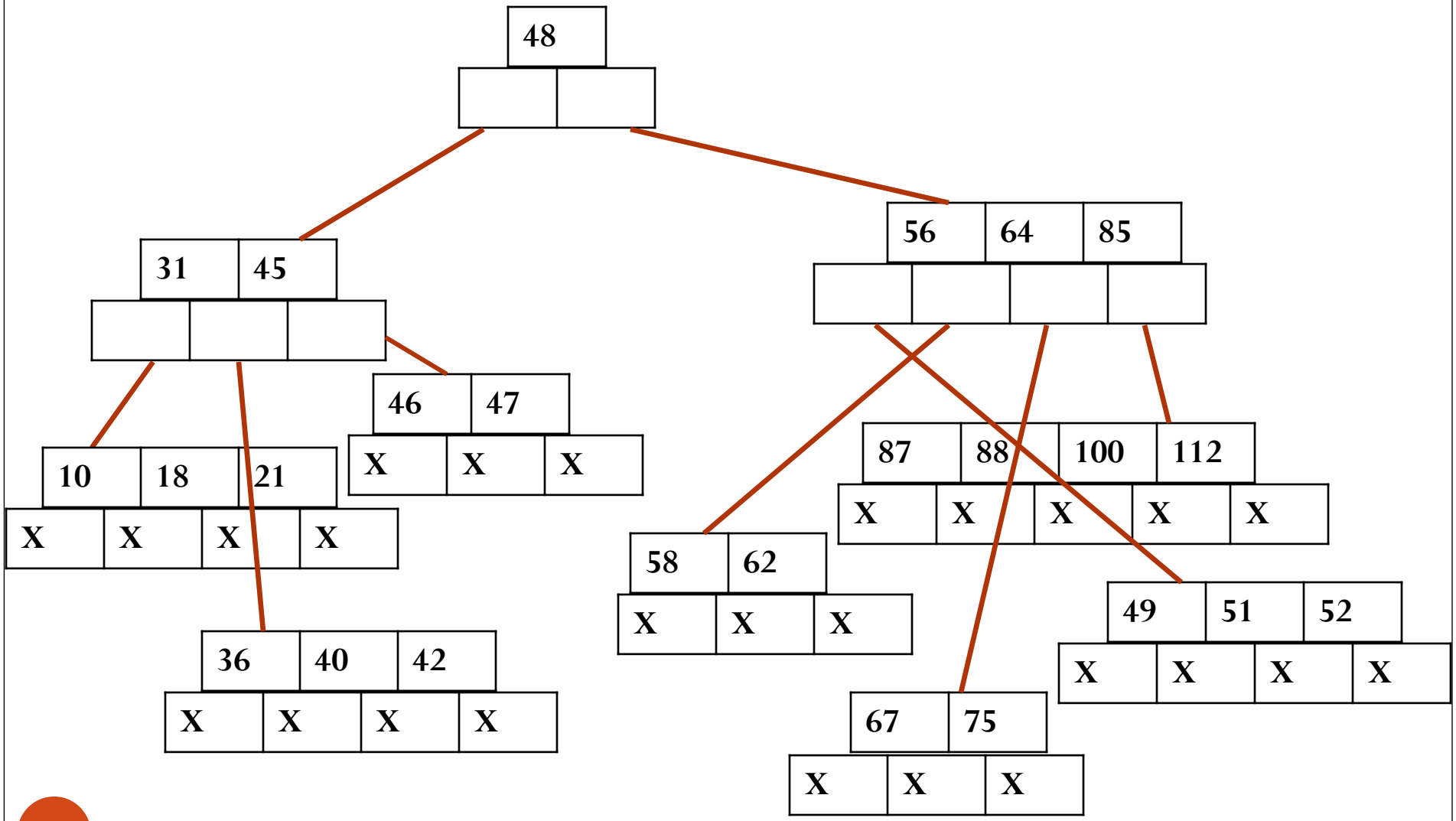
--replace 198 by 262

B Trees

B tree is a balanced **m-way** search tree

- A B tree of order m, if non empty, is an m-way search tree in which
  i. the root has at least **two** child pointers and at most **m** child pointers
  ii. nodes except the root have at least $\lceil m/2 \rceil$ child pointers and at most **m** child pointers
  iii. all leaf nodes are on the same level

B Tree of order 5



Searching a B Tree

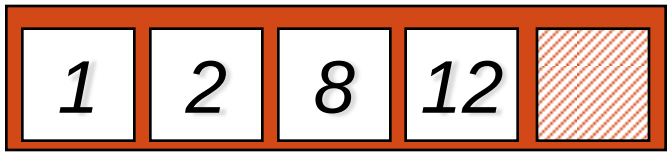
- Searching for a key in a B-tree is similar to the one on an m-way search tree.
- The number of accesses depends on the height **h** of the B-tree

Insertion in a B-Tree

1. Attempt to insert the new key into a leaf
2. If this would result in that leaf becoming too big, split the leaf into two, promoting the middle key to the leaf's parent
3. If this would result in the parent becoming too big, split the parent into two, promoting the middle key
4. This strategy might have to be repeated all the way to the top
5. If necessary, the root is split in two and the middle key is promoted to a new root, making the tree one level higher

Constructing a B-tree

- Suppose we start with an empty B-tree and keys arrive in the following order:1 12 8 2 25 6 14 28 17 7 52 16 48 68 3 26 29 53 55 45
- We want to construct a B-tree of order 5
- The first four items go into the root:

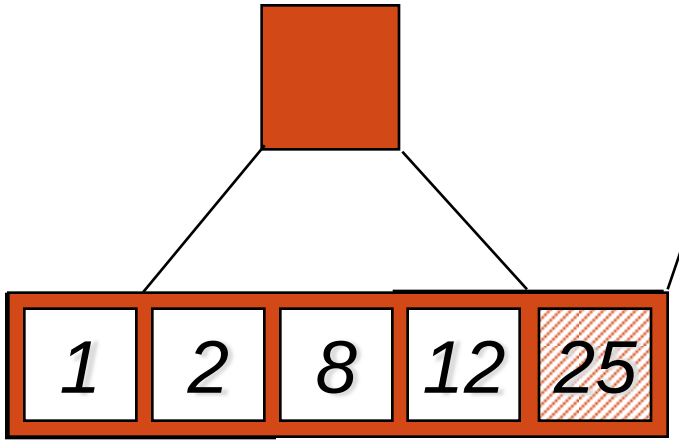


- To put the fifth item in the root would over-fill it
- Therefore, when 25 arrives, pick the middle key to make a new root

1
12
8
2
25
6
14
28
17
7
52
16
48
68
3
26
29
53
55
45

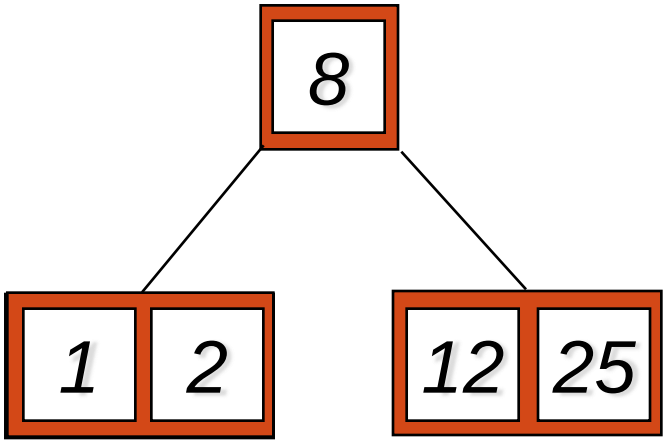
Constructing a B-tree

Add 25 to the tree

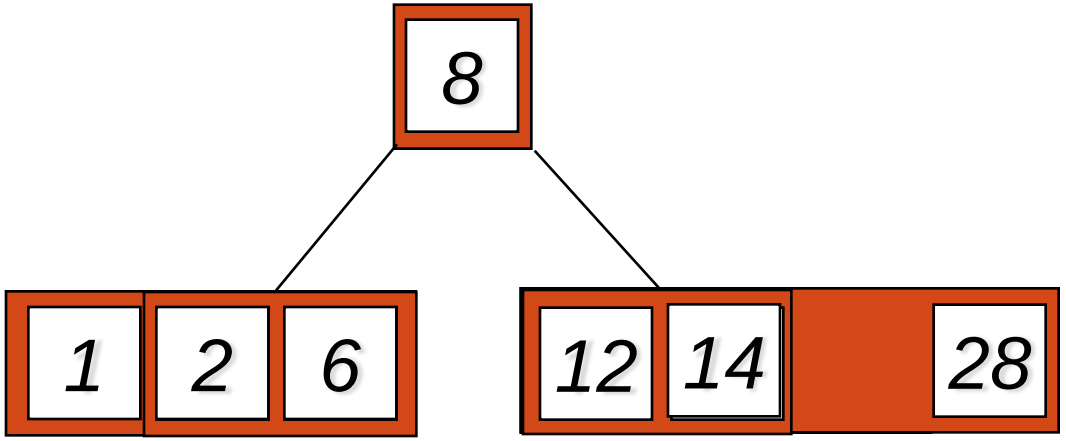


Exceeds Order.
Promote middle and split.

Constructing a B-tree (contd.)



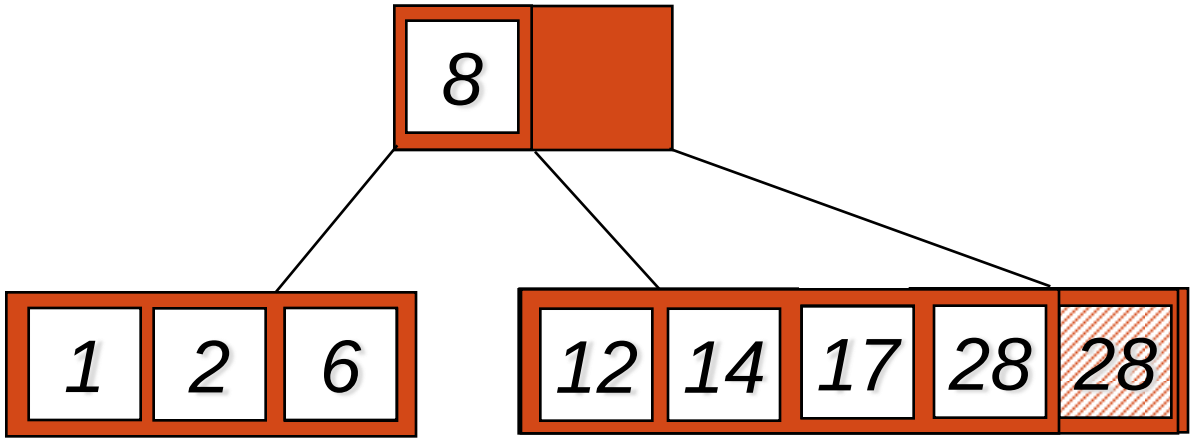
6, 14, 28 get added to the leaf nodes:



1
12
8
2
25
6
14
28
17
7
52
16
48
68
3
26
29
53
55
45

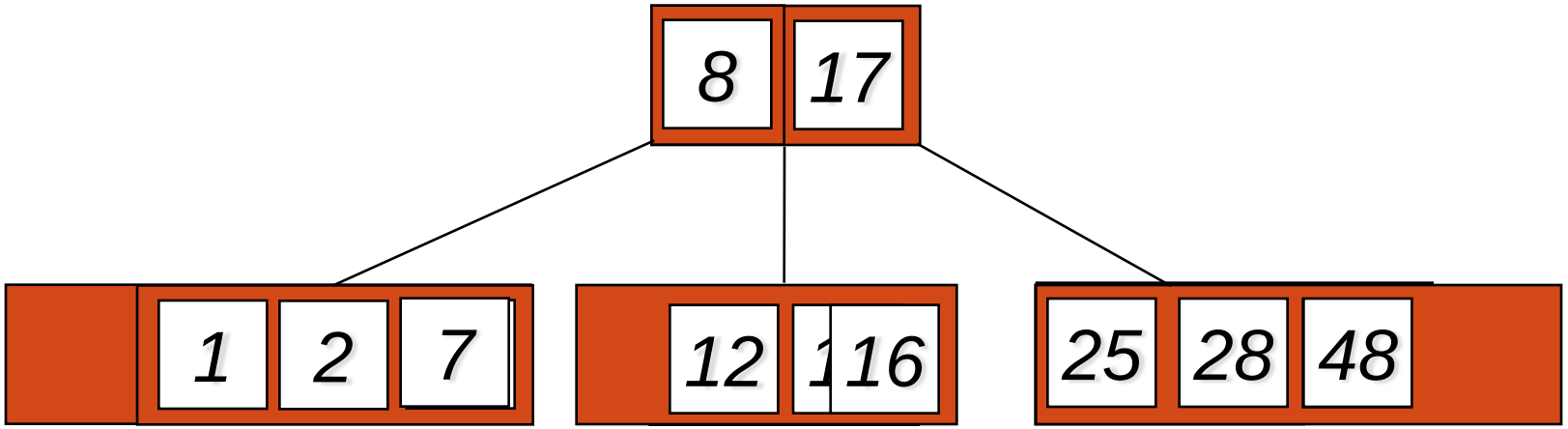
Constructing a B-tree (contd.)

Adding 17 to the right leaf node would over-fill it, so we take the middle key, promote it (to the root) and split the leaf



Constructing a B-tree (contd.)

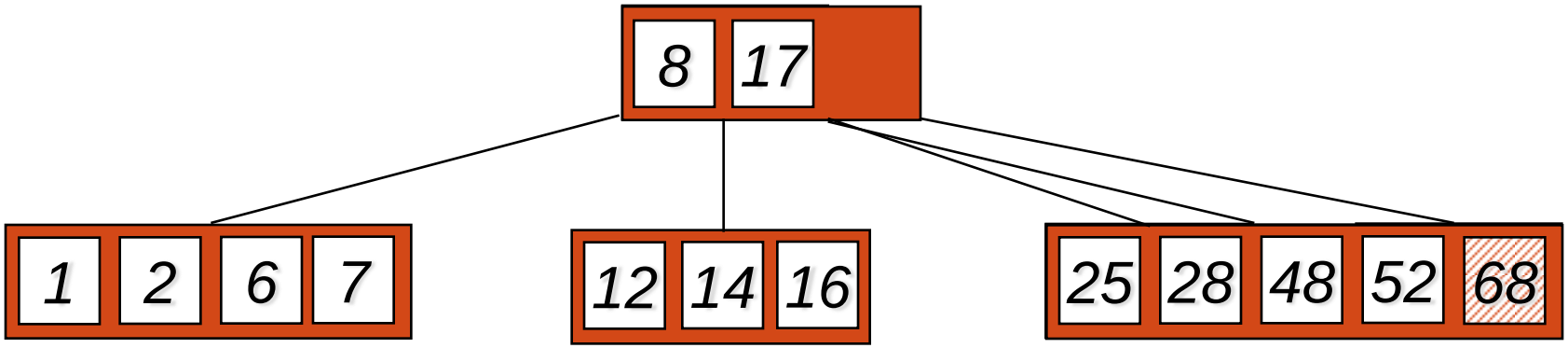
7, 52, 16, 48 get added to the leaf nodes



1
12
8
2
25
6
14
28
17
7
52
16
48
68
3
26
29
53
55
45

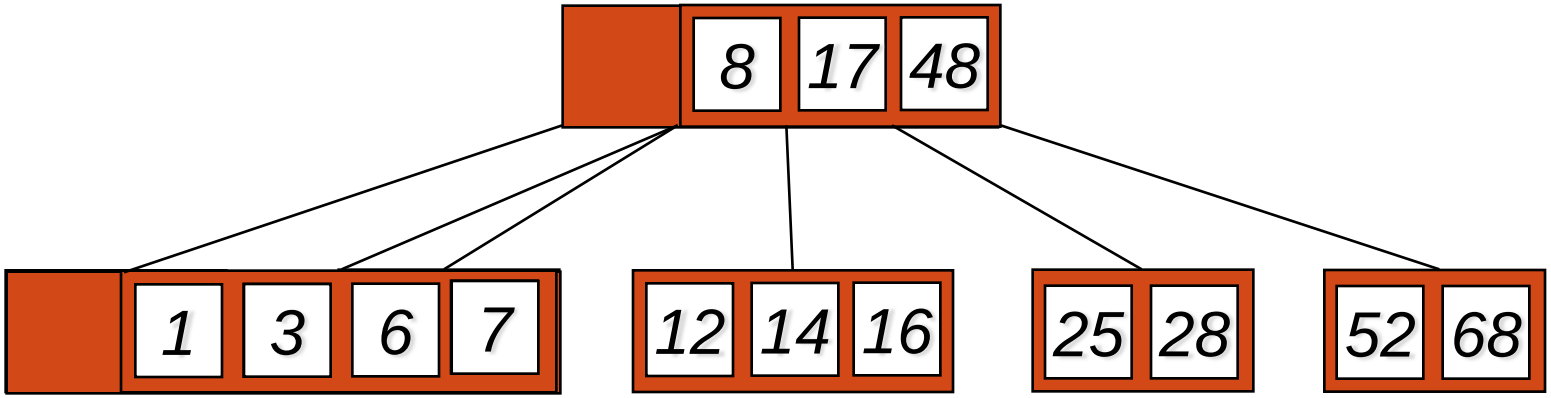
Constructing a B-tree (contd.)

Adding 68 causes us to split the right most leaf, promoting 48 to the root



Constructing a B-tree (contd.)

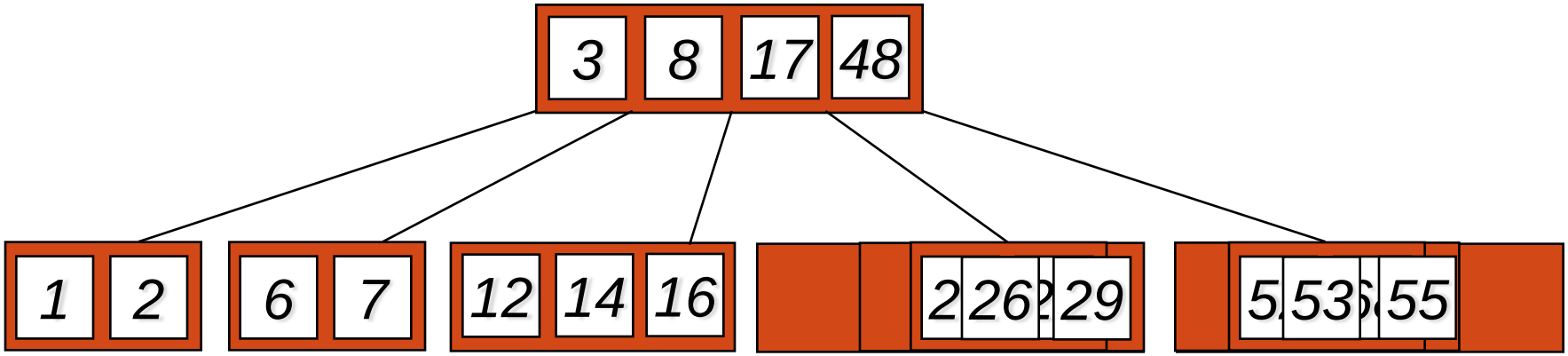
Adding 3 causes us to split the left most leaf



1
12
8
2
25
6
14
28
17
7
52
16
48
68
3
26
29
53
55
45

Constructing a B-tree (contd.)

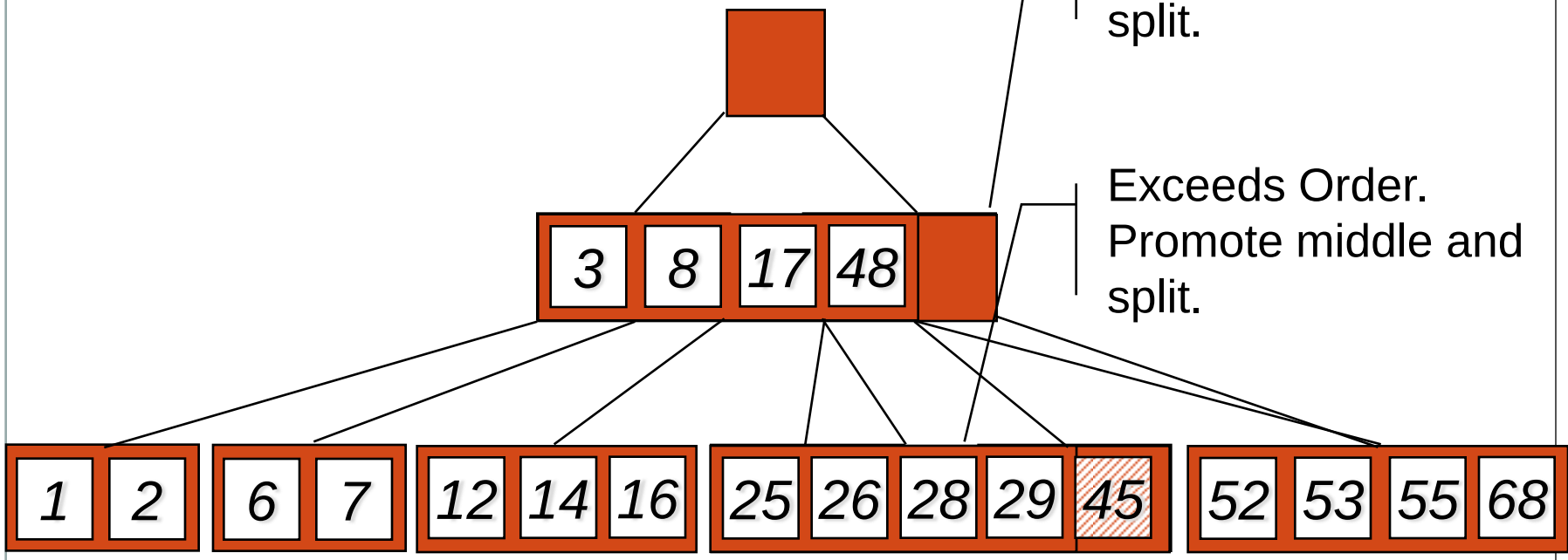
Add 26, 29, 53, 55 then go into the leaves



1
12
8
2
25
6
14
28
17
7
52
16
48
68
3
26
29
53
55
45

Constructing a B-tree (contd.)

Add 45 increases the trees level

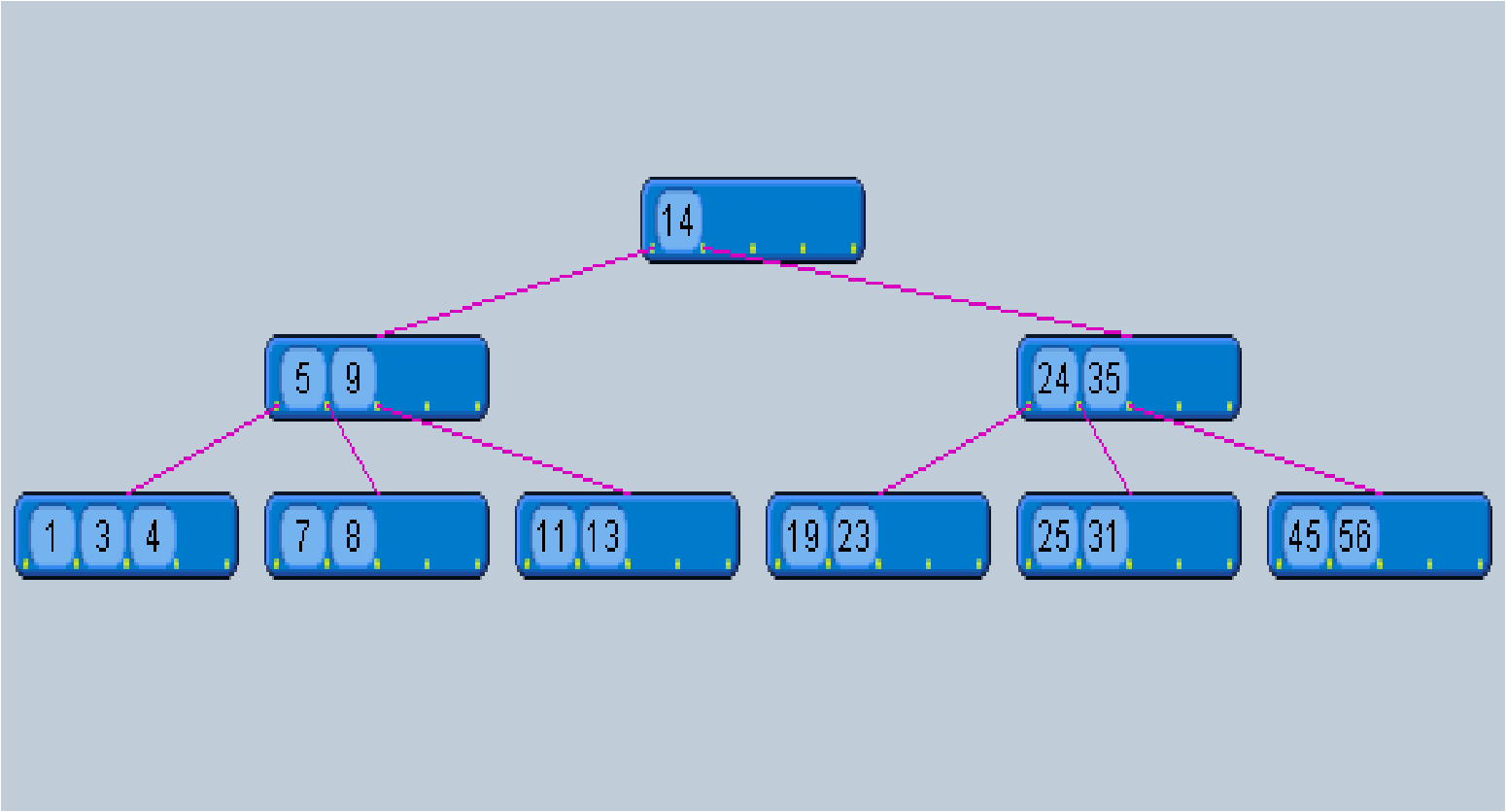


Exercise in Inserting a B-Tree

- Insert the following keys to a 5-way B-tree:

3, 7, 9, 23, 45, 1, 5, 14, 25, 24, 13, 11, 8, 19, 4, 31, 35, 56

Answer to Exercise



Delete from a B-tree

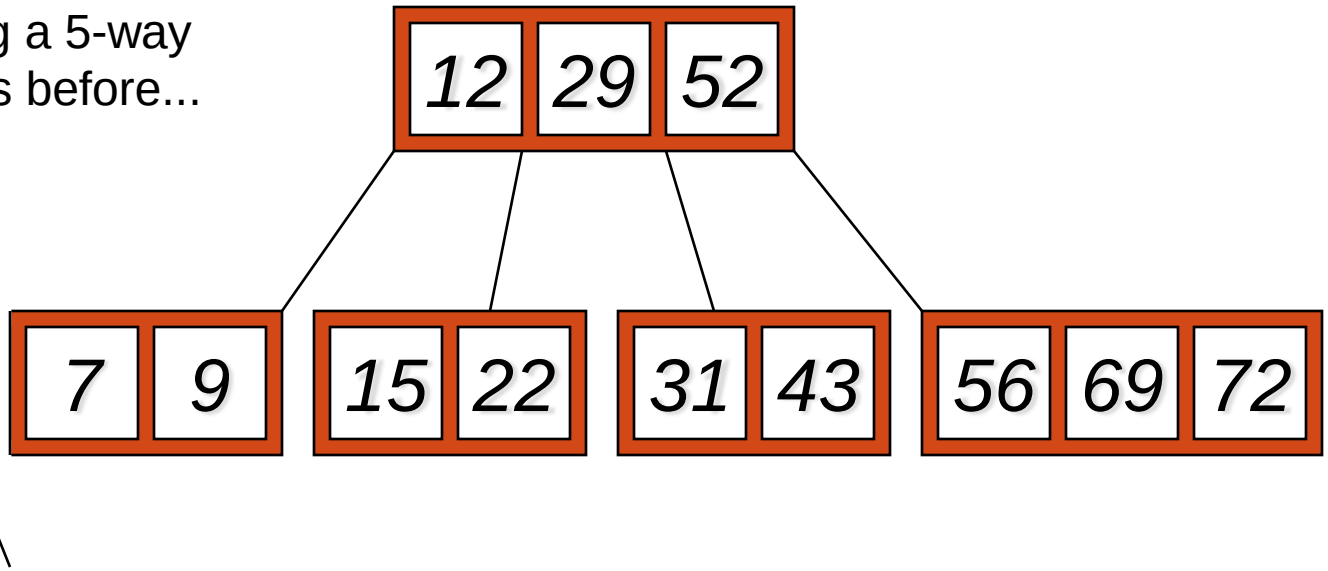
1. If the key is already in a leaf node, and removing it doesn't cause that leaf node to have too few keys, then simply remove the key to be deleted.
2. If the key is *not* in a leaf then it is guaranteed (by the nature of a B-tree) that its predecessor or successor will be in a leaf -- in this case can we delete the key and promote the predecessor or successor key to the non-leaf deleted key's position.

Removal from a B-tree (2)

- If (1) or (2) lead to a leaf node containing less than the minimum number of keys then we have to look at the siblings immediately adjacent to the leaf in question:
  - 3: if one of them has more than the min' number of keys then we can promote one of its keys to the parent and take the parent key into our lacking leaf
  - 4: if neither of them has more than the min' number of keys then the lacking leaf and one of its neighbours can be combined with their shared parent (the opposite of promoting a key) and the new leaf will have the correct number of keys; if this step leave the parent with too few keys then we repeat the process up to the root itself, if required

Type #1: Simple leaf deletion

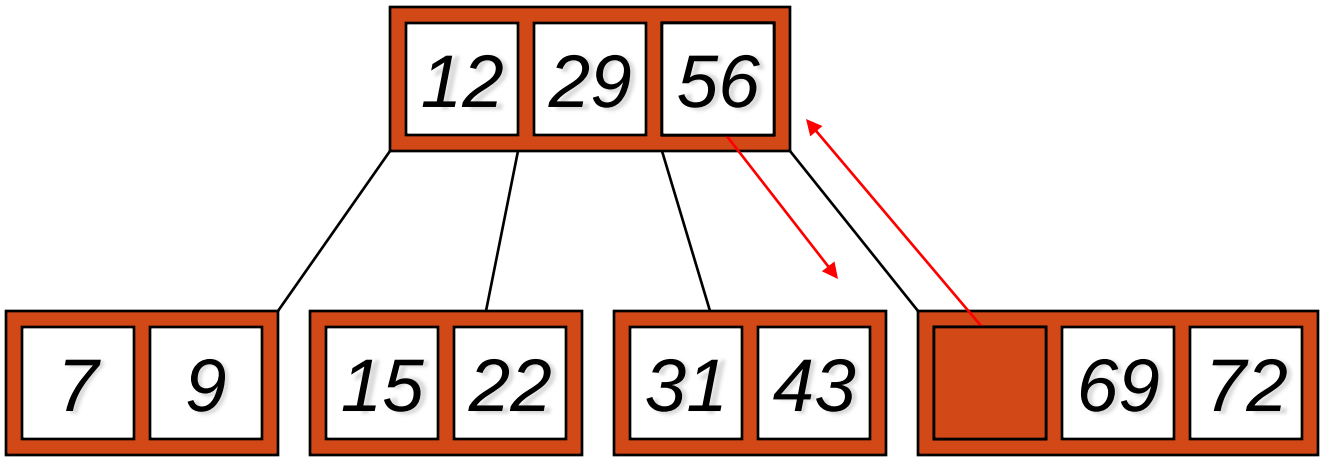
Assuming a 5-way B-Tree, as before...



Delete 2: Since there are enough keys in the node, just delete it

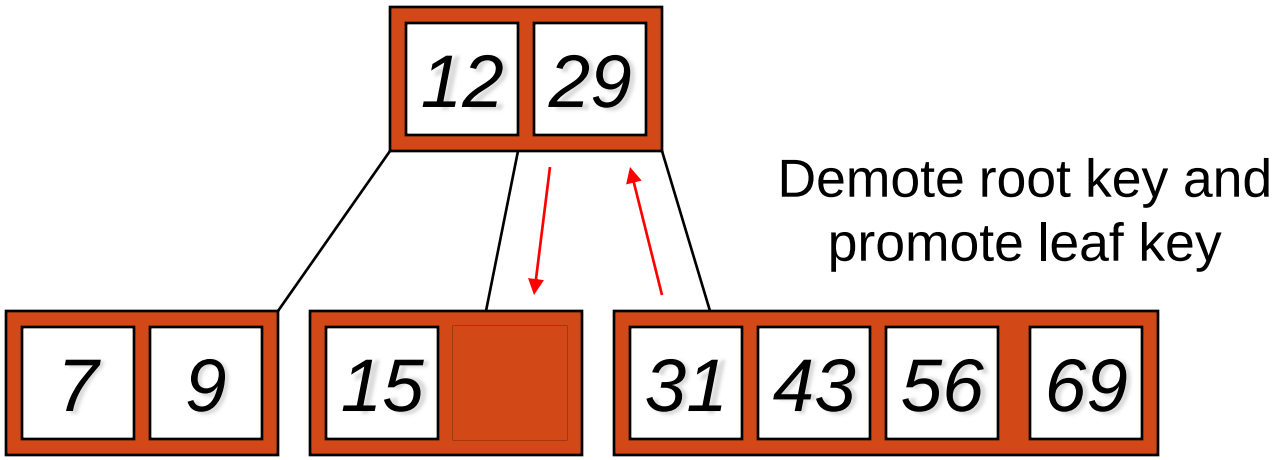
Note when printed: this slide is animated

Type #2: Simple non-leaf deletion



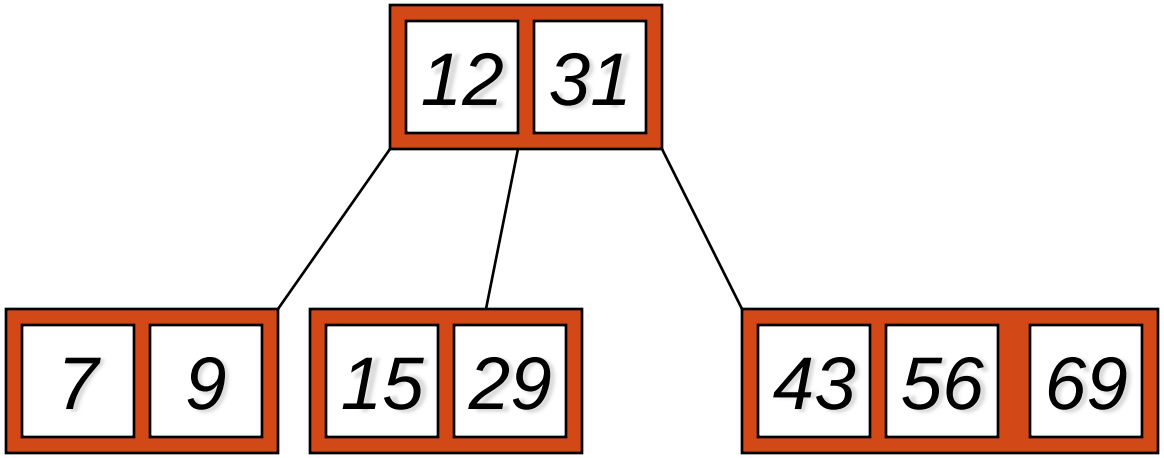
Note when printed: this slide is animated

Type #3: Enough siblings



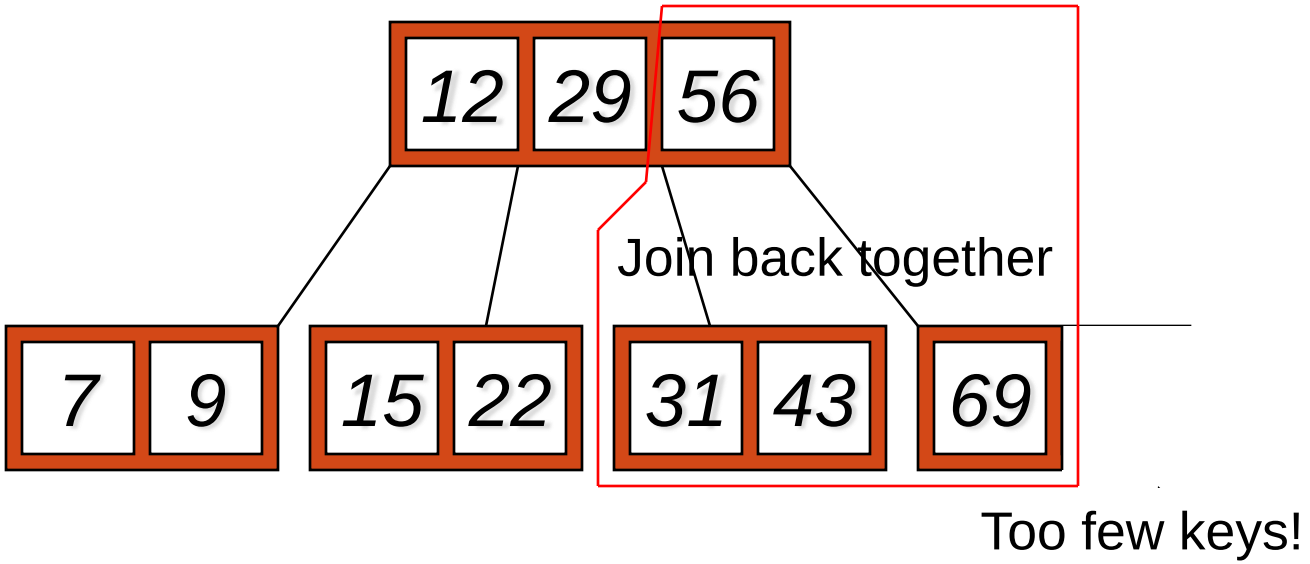
Note when printed: this slide is animated

Type #3: Enough siblings



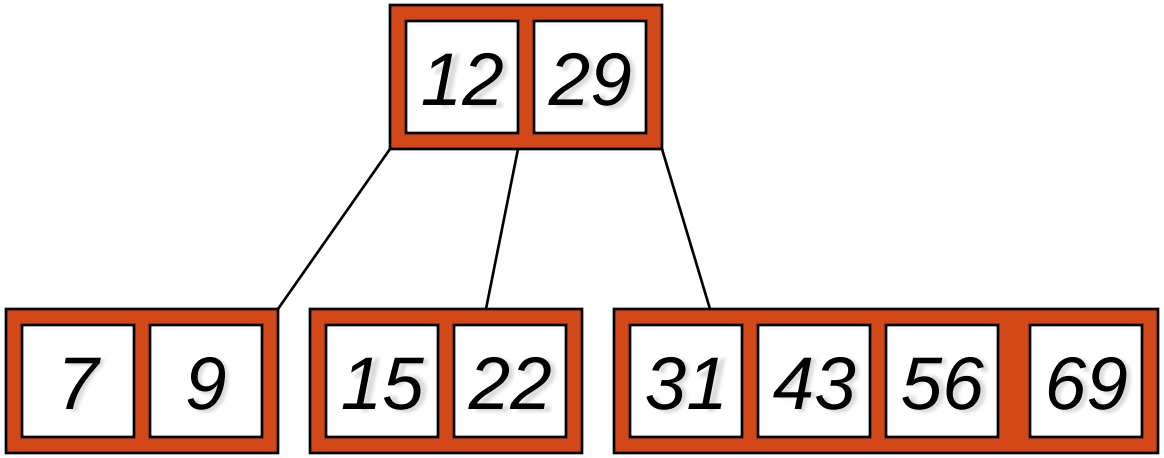
Note when printed: this slide is animated

Type #4: Too few keys in node and its siblings



Note when printed: this slide is animated

Type #4: Too few keys in node and its siblings



Note when printed: this slide is animated

Exercise in Removal from a B-Tree

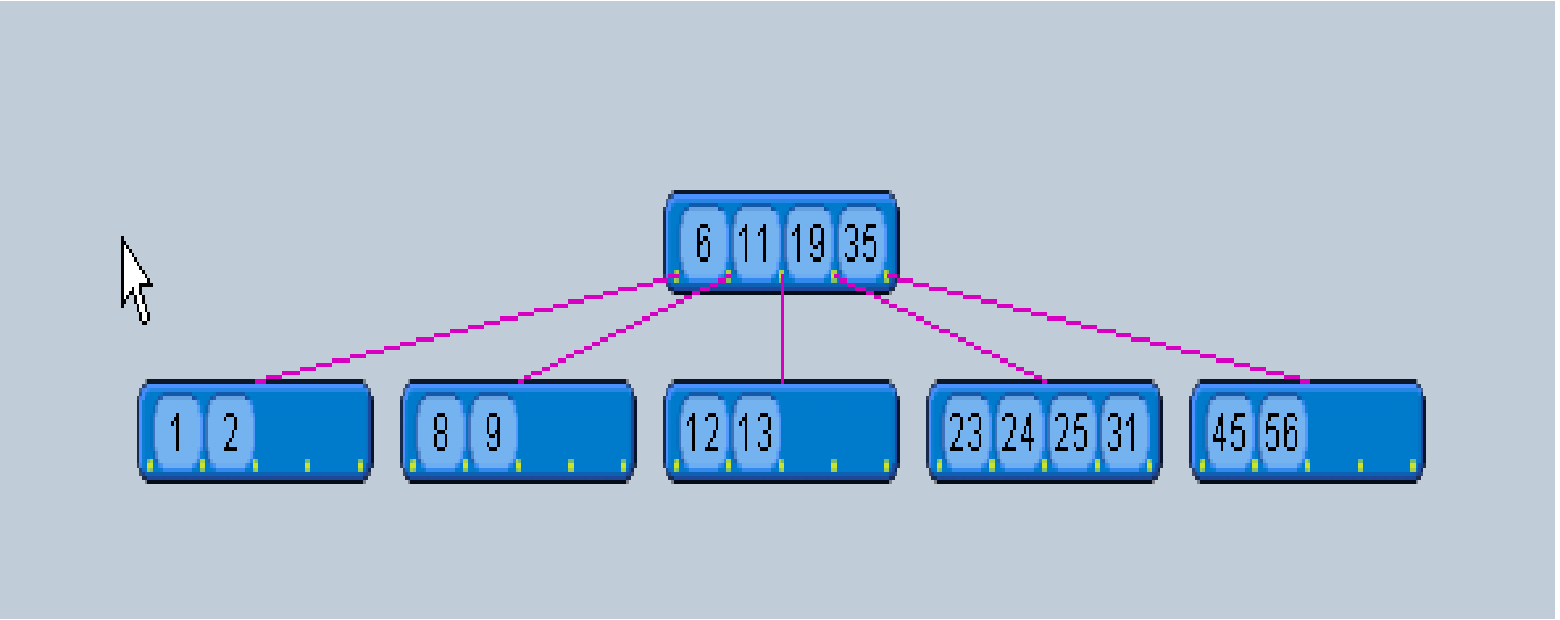
- Given 5-way B-tree created by these data (last exercise):

3, 7, 9, 23, 45, 1, 5, 14, 25, 24, 13, 11, 8, 19, 4, 31, 35, 56

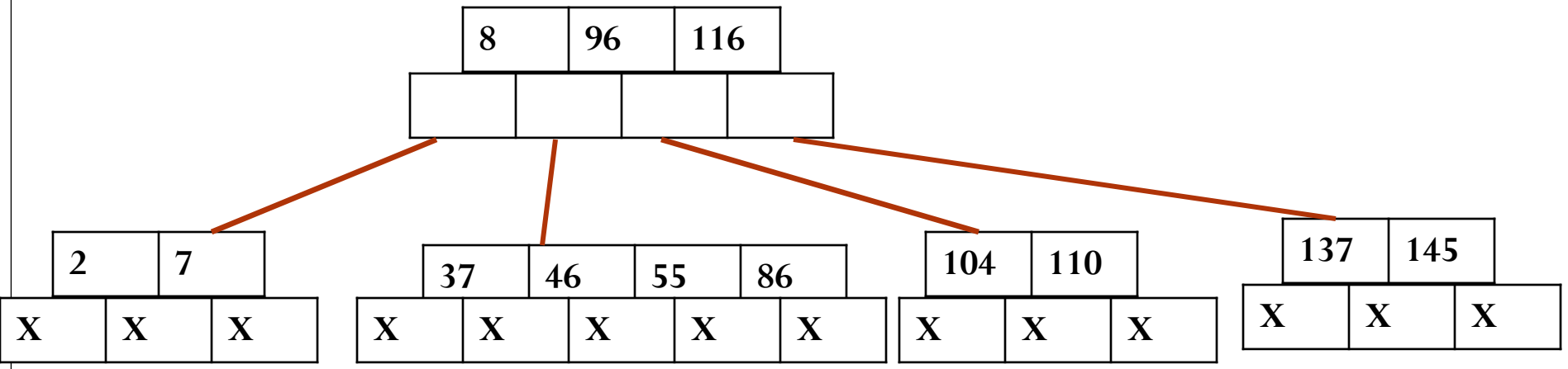
- Add these further keys: 2, 6,12

- Delete these keys: 4, 5, 7, 3, 14

Answer to Exercise

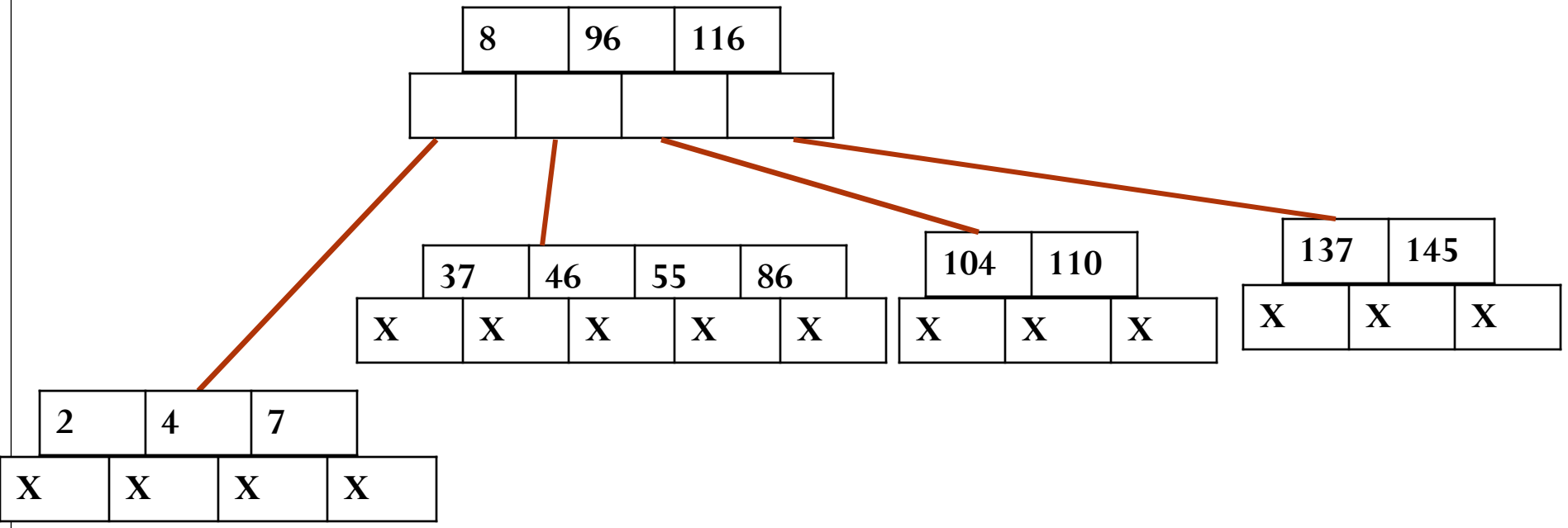


5-Way B Tree (insertion examples)



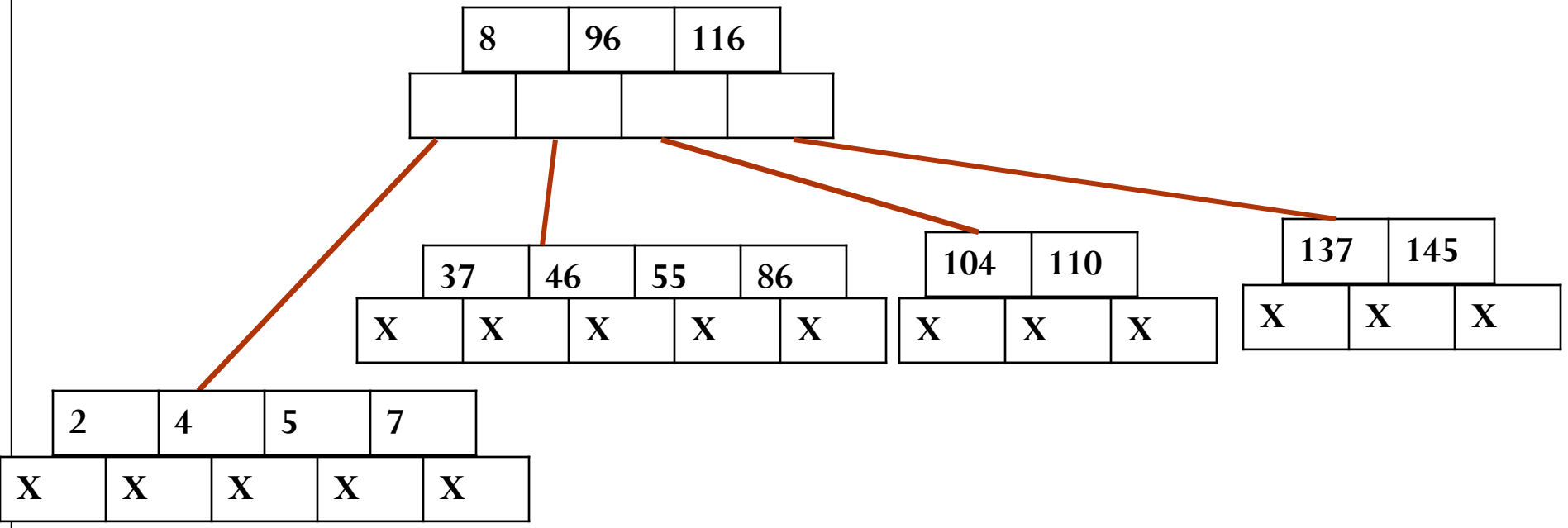
Insert 4, 5, 58, 6 in the order

5-Way B Tree (insertion examples)



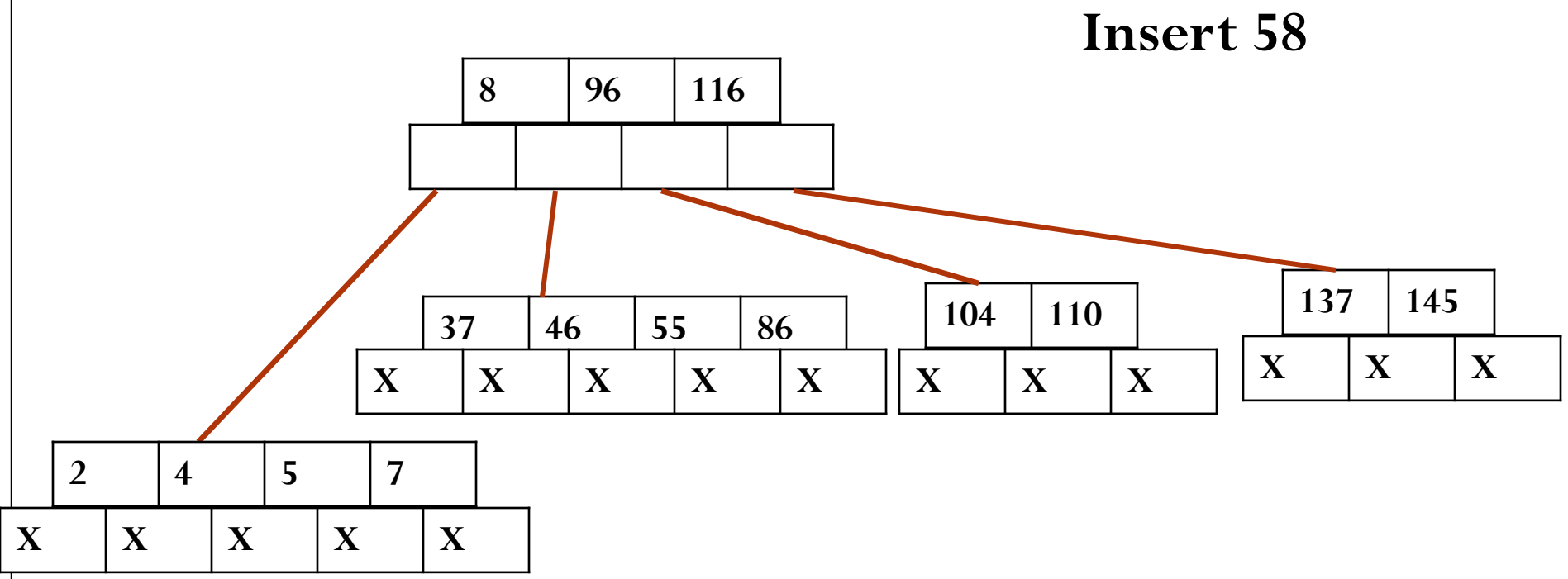
Search tree after inserting 4

5-Way B Tree (insertion examples)



Search tree after inserting 4, 5

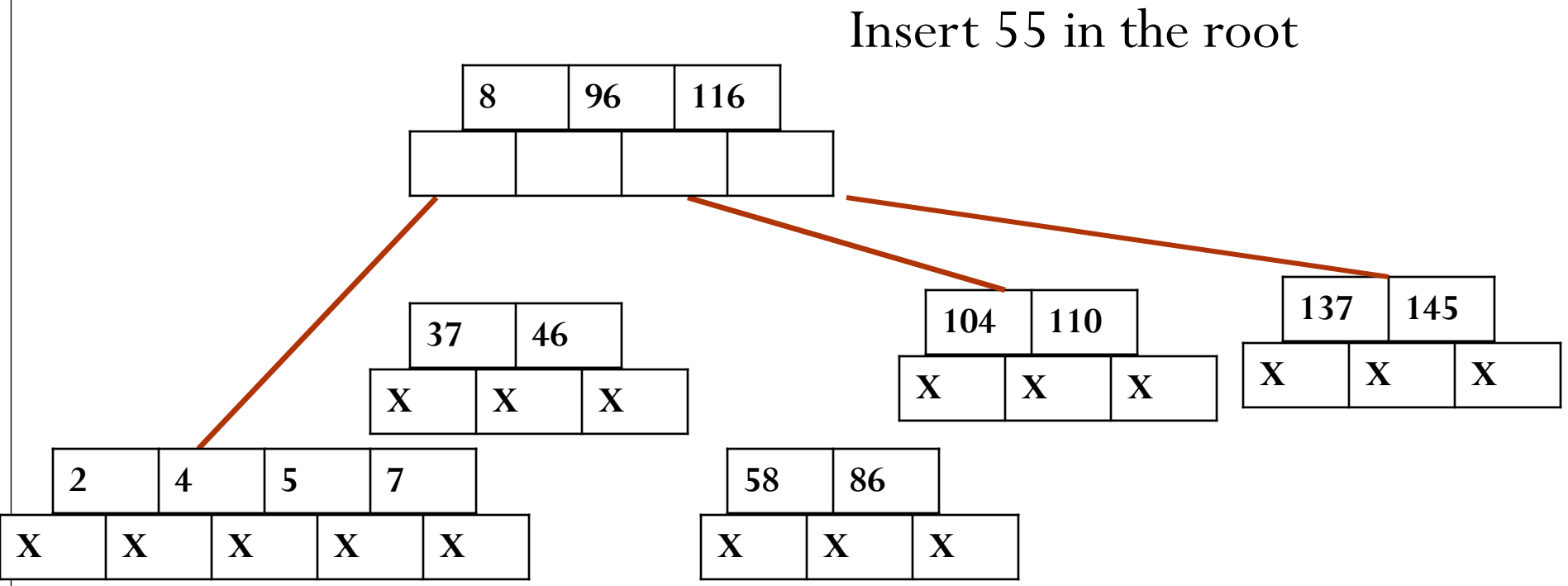
5-Way B Tree (insertion examples)



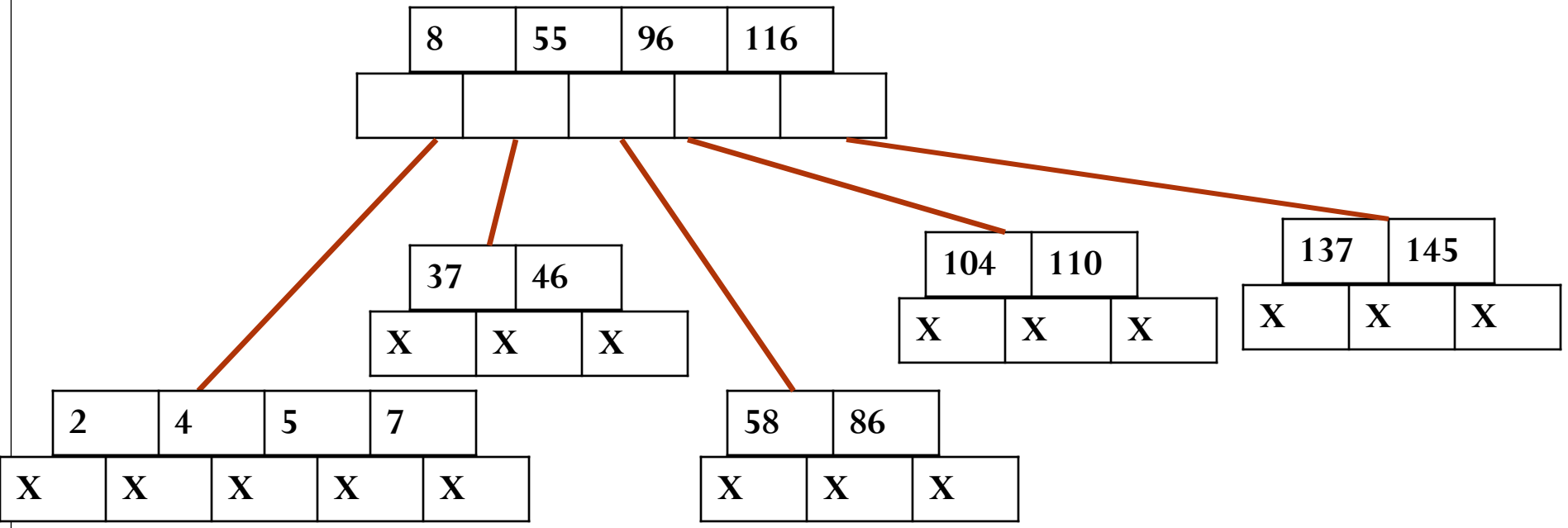
37,46,55,58,86

Split the node at its median into two node, pushing the median element up by one level

5-Way B Tree (insertion examples)

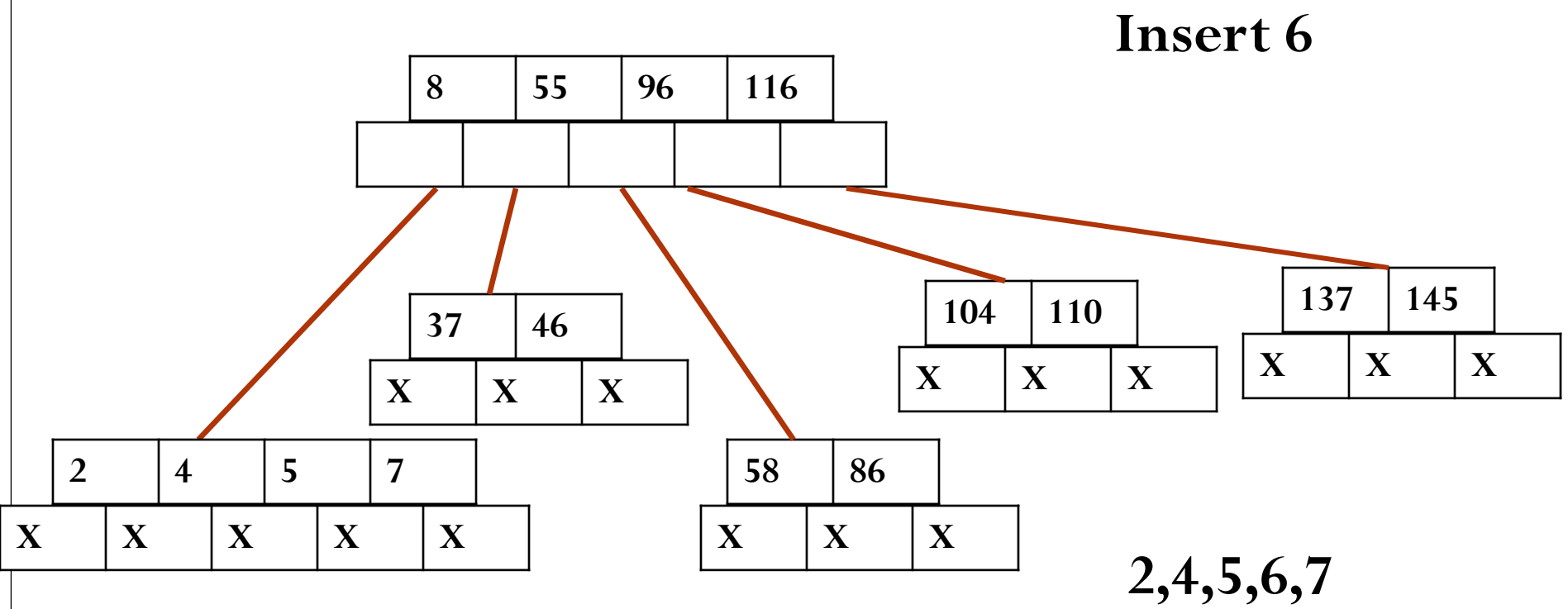


5-Way B Tree (insertion examples)



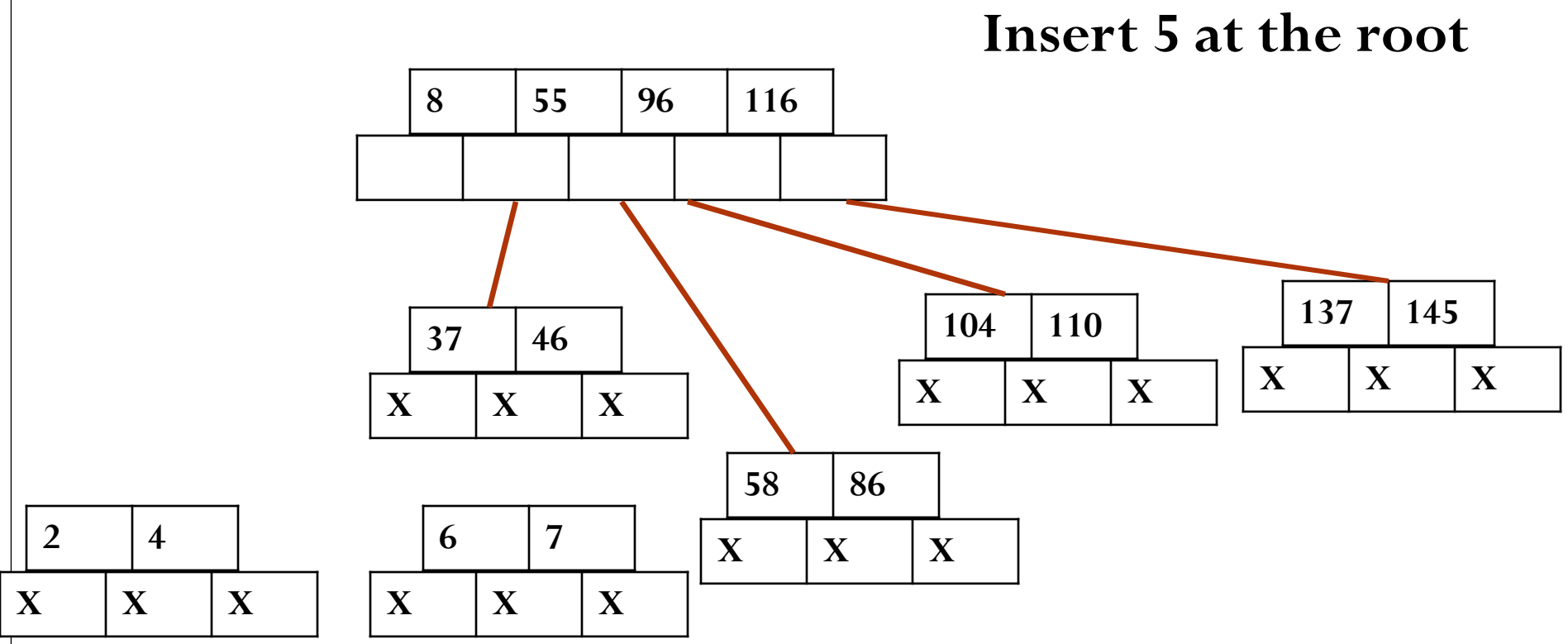
Search tree after inserting 4, 5, 58

5-Way B Tree (insertion examples)

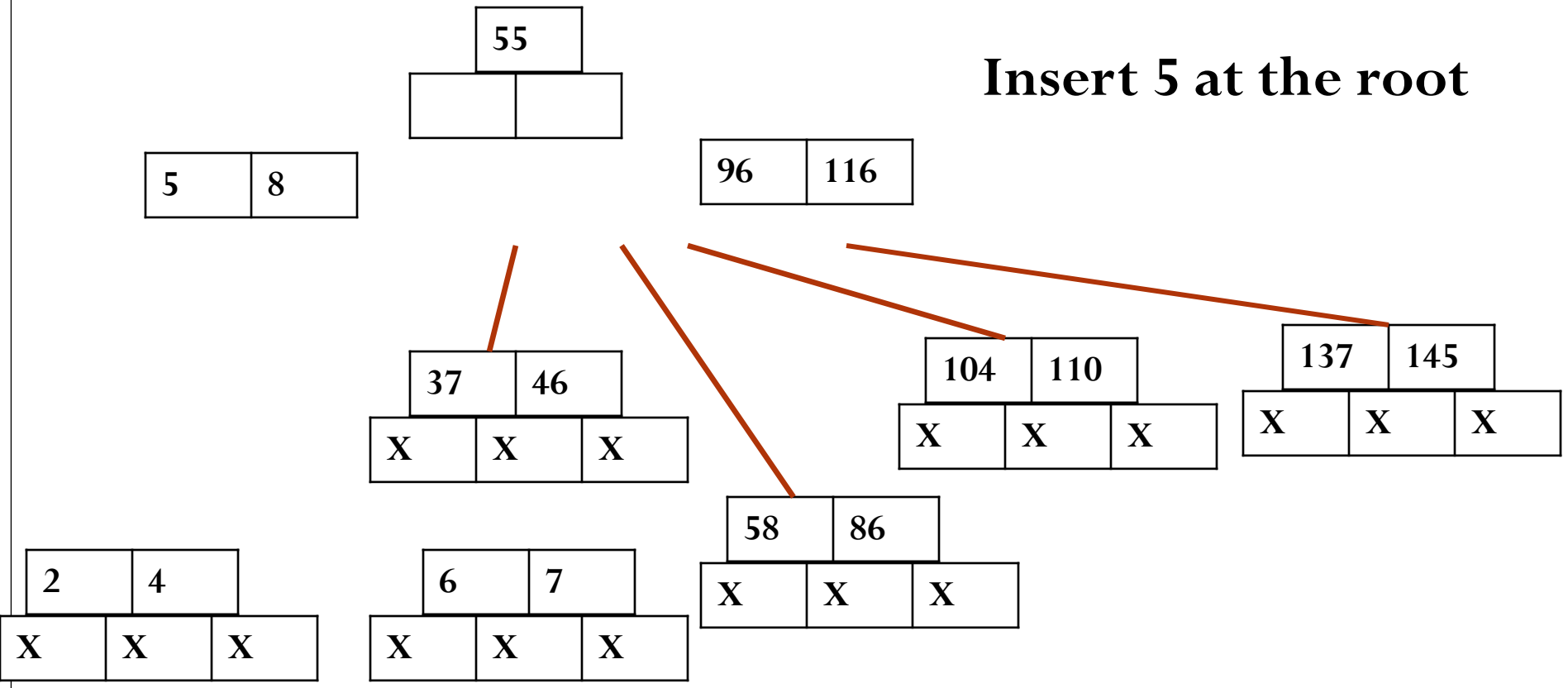


Split the node at its median into two node, pushing the median element up by one level

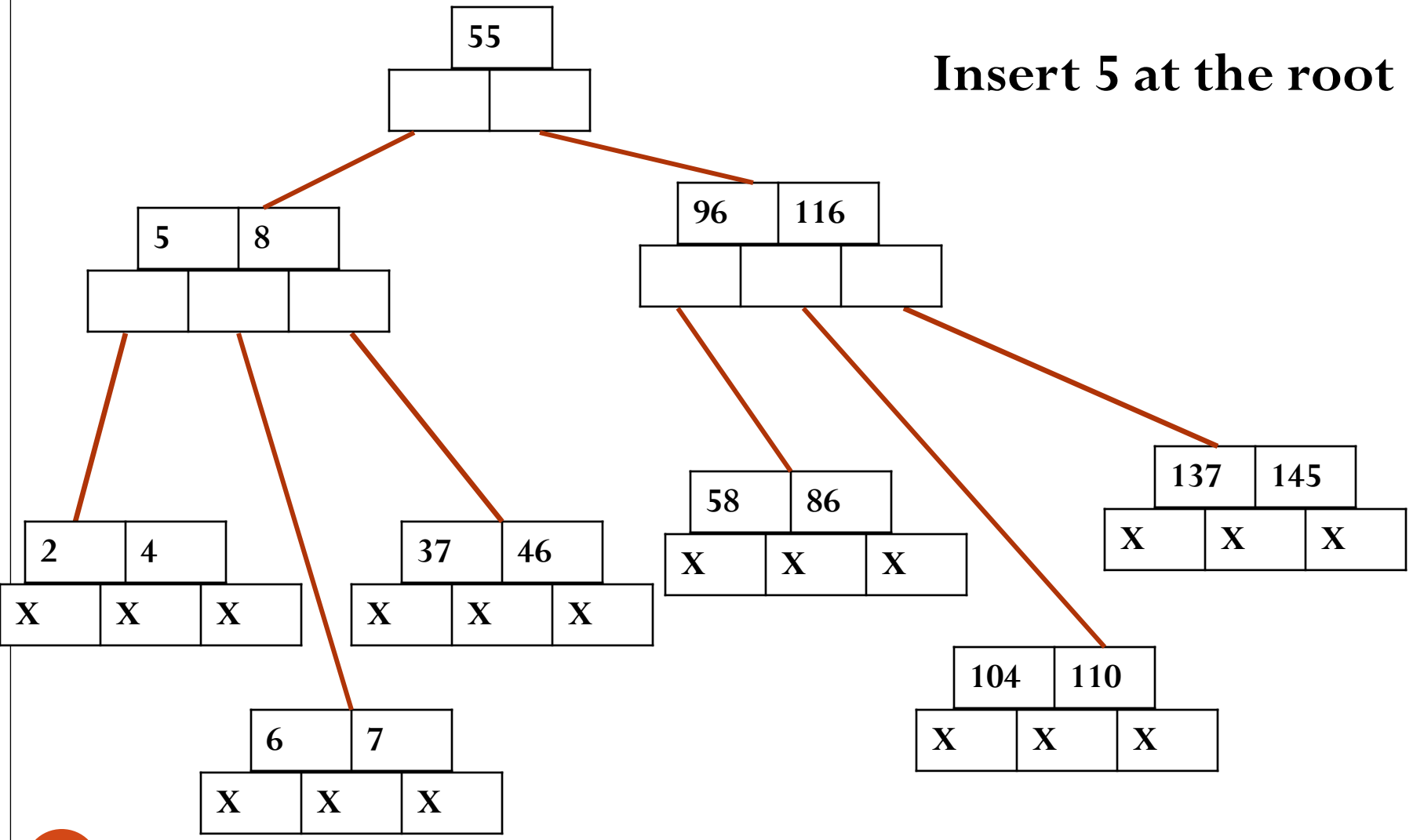
5-Way B Tree (insertion examples)



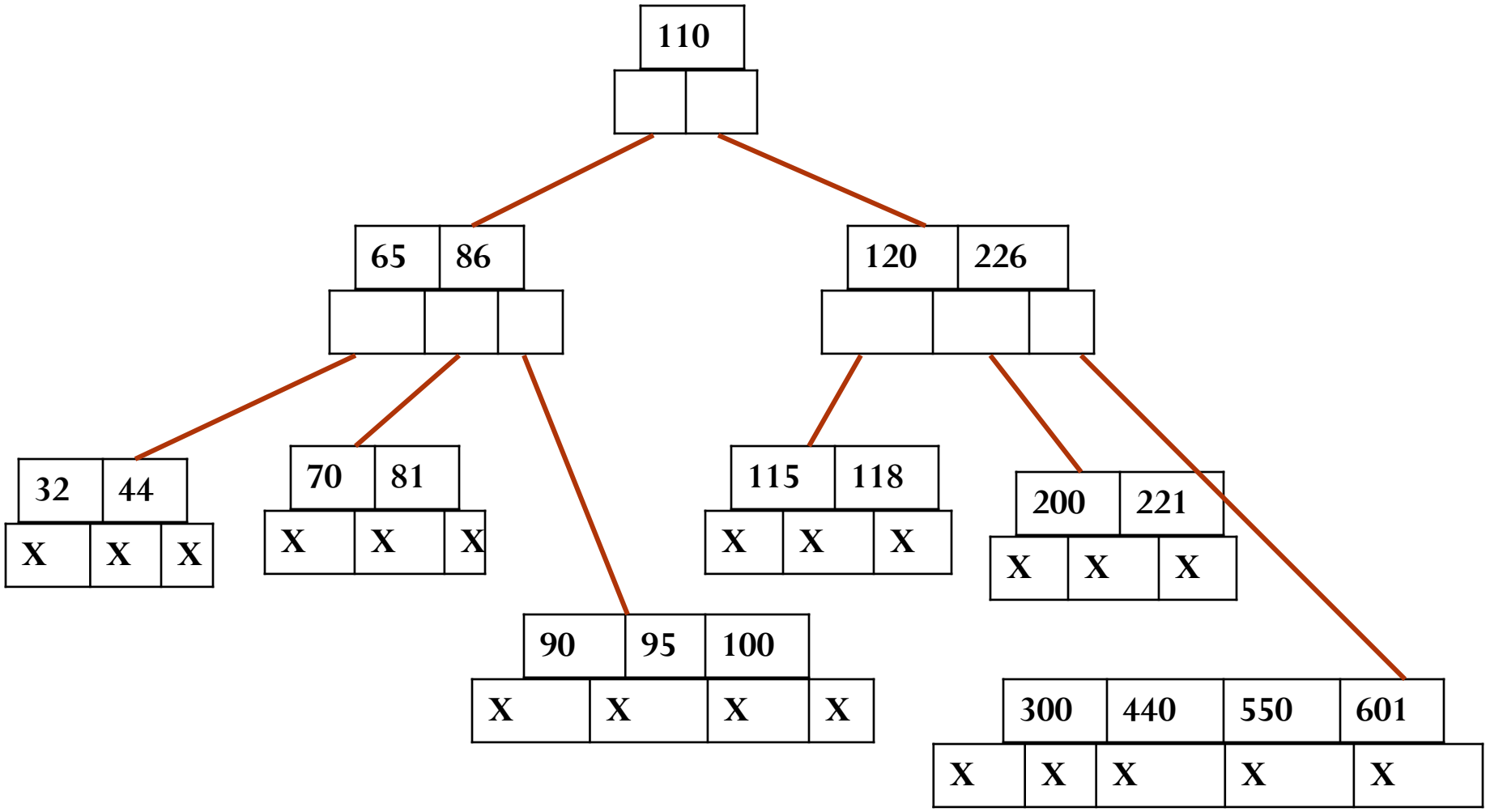
5-Way B Tree (insertion examples)



5-Way B Tree (insertion examples)

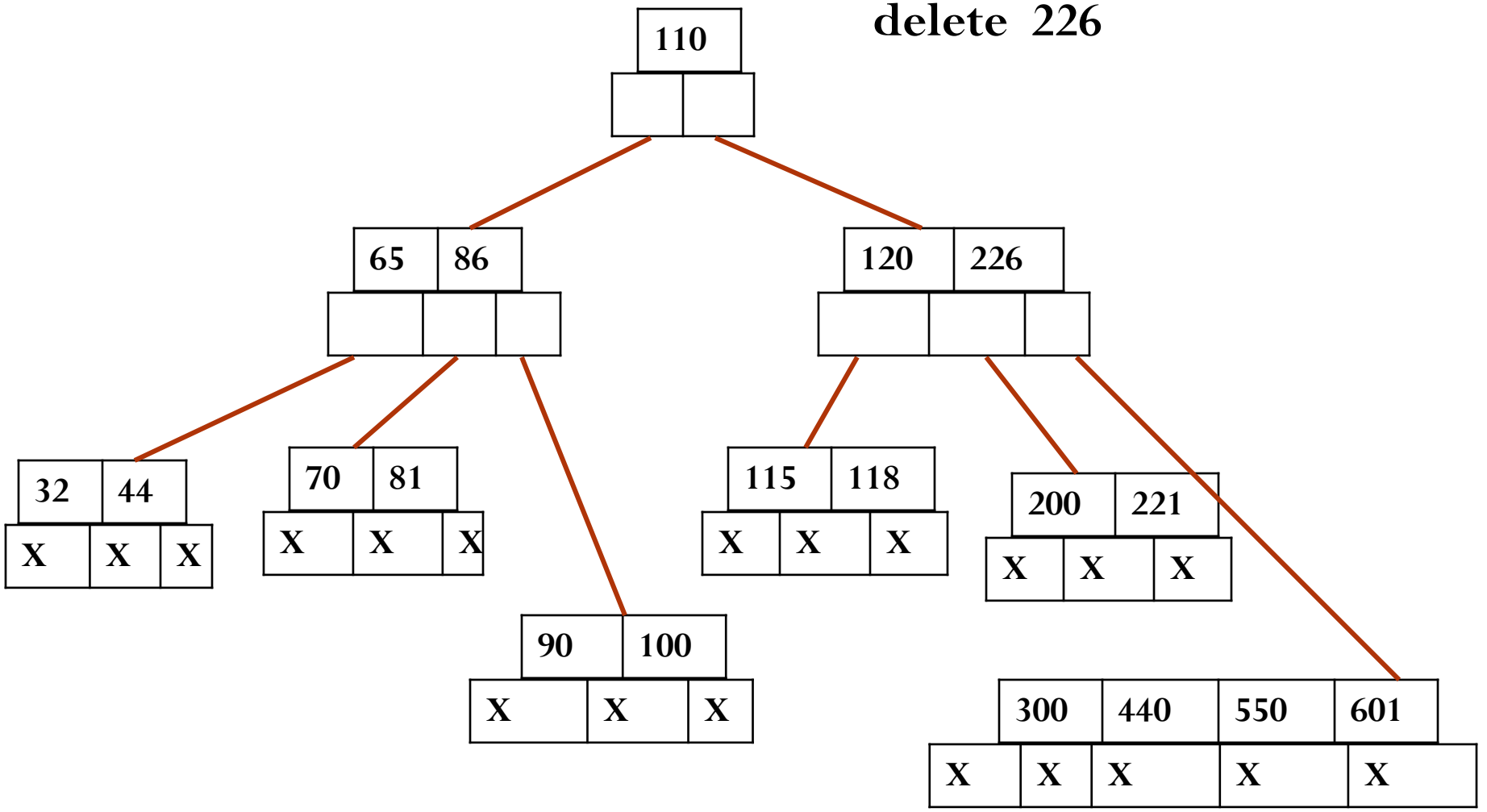


B-tree of Order 5 (deletion examples)



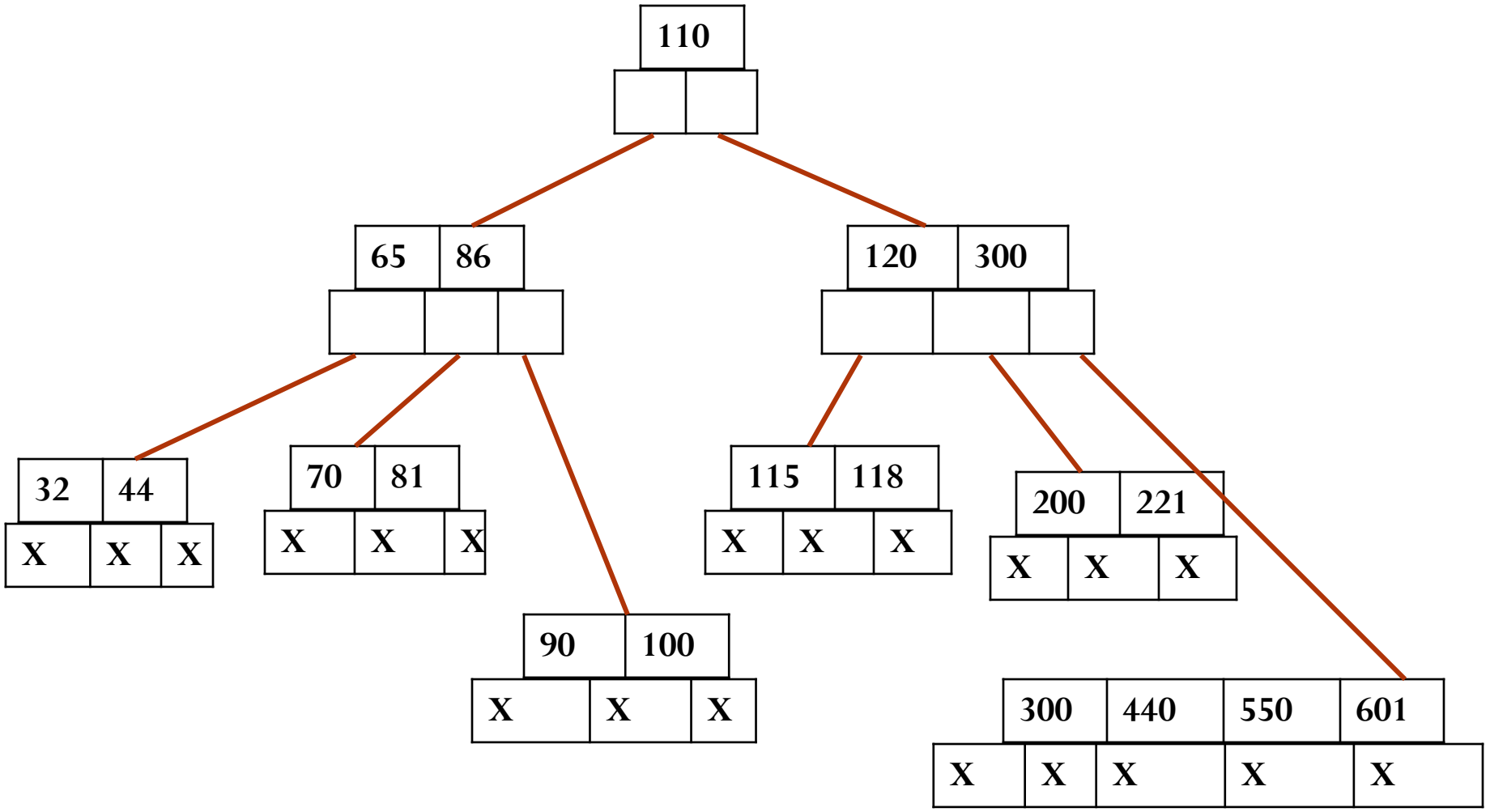
Delete 95, 226, 221, 70

B-tree of Order 5

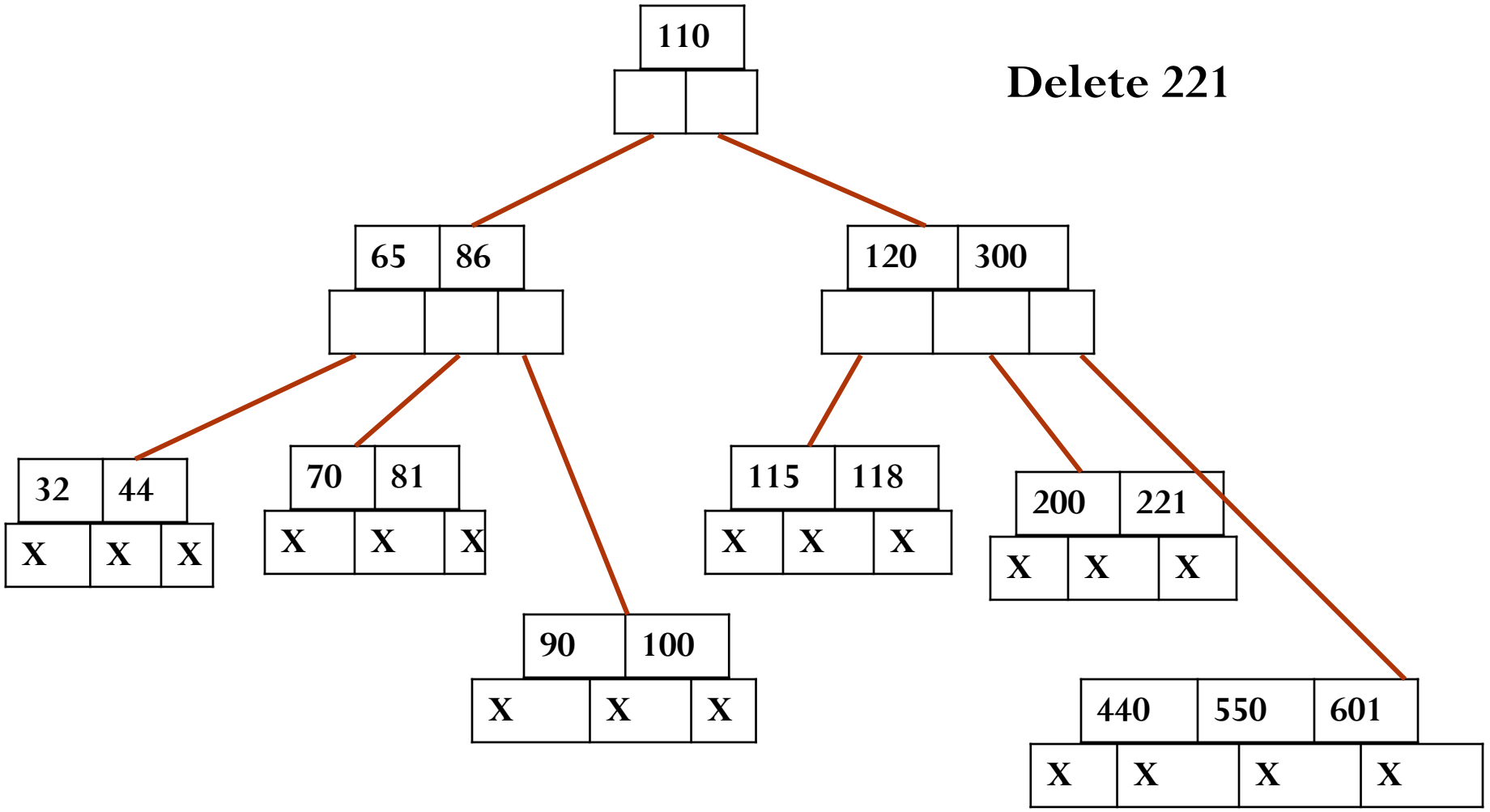


B-tree after deleting 95

B-tree of Order 5

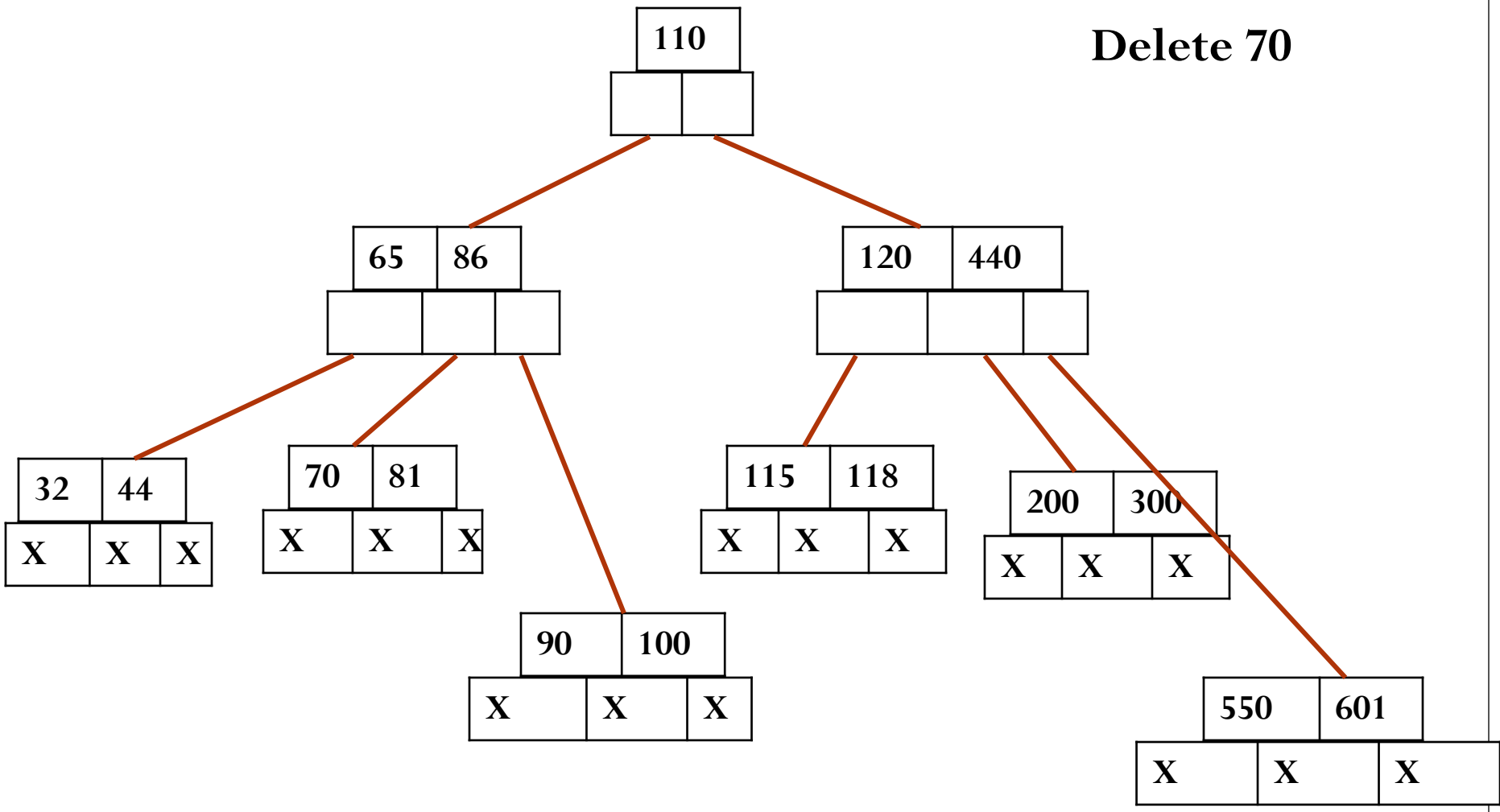


B-tree of Order 5



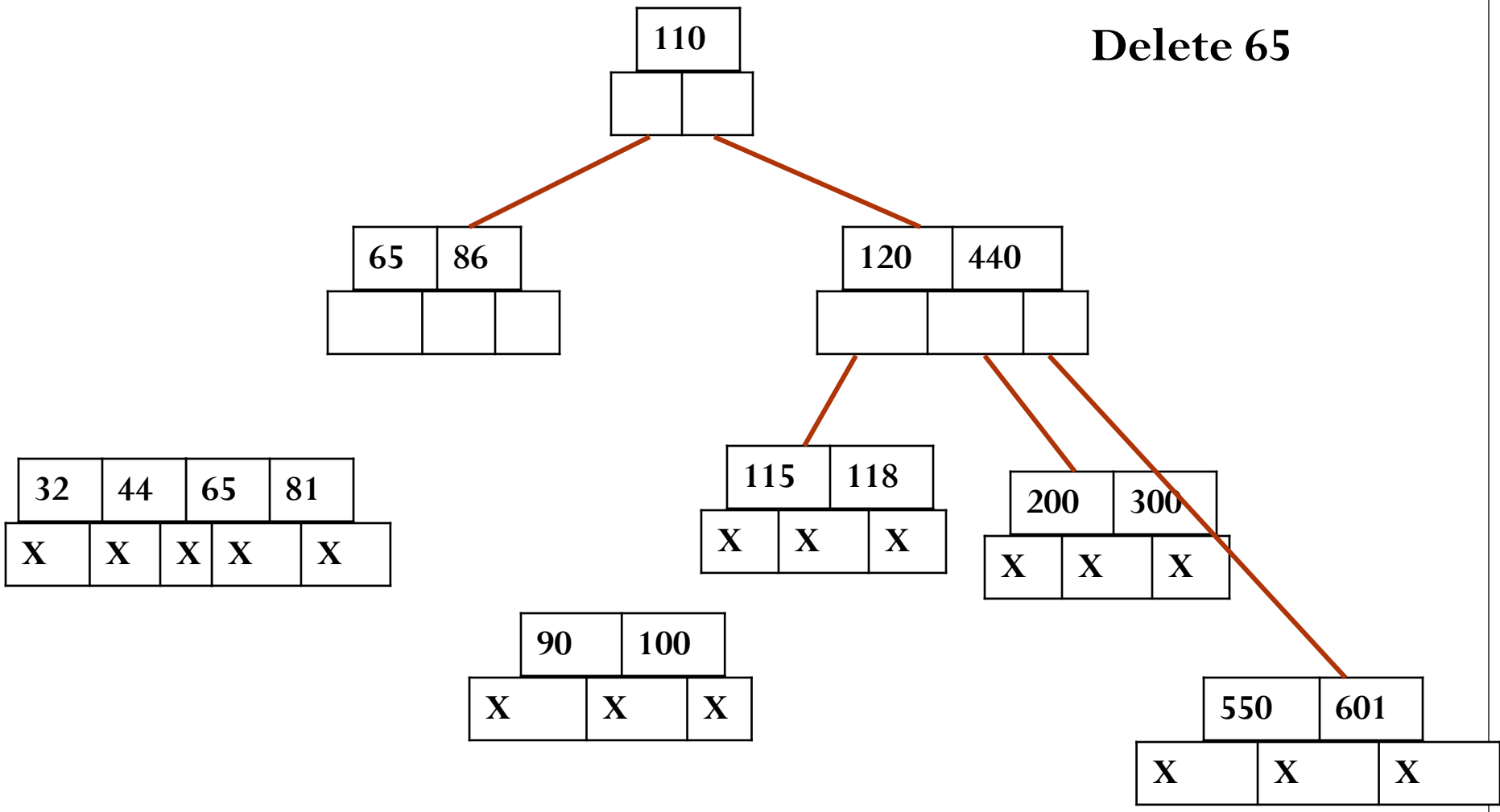
B-tree after deleting 95, 226

B-tree of Order 5

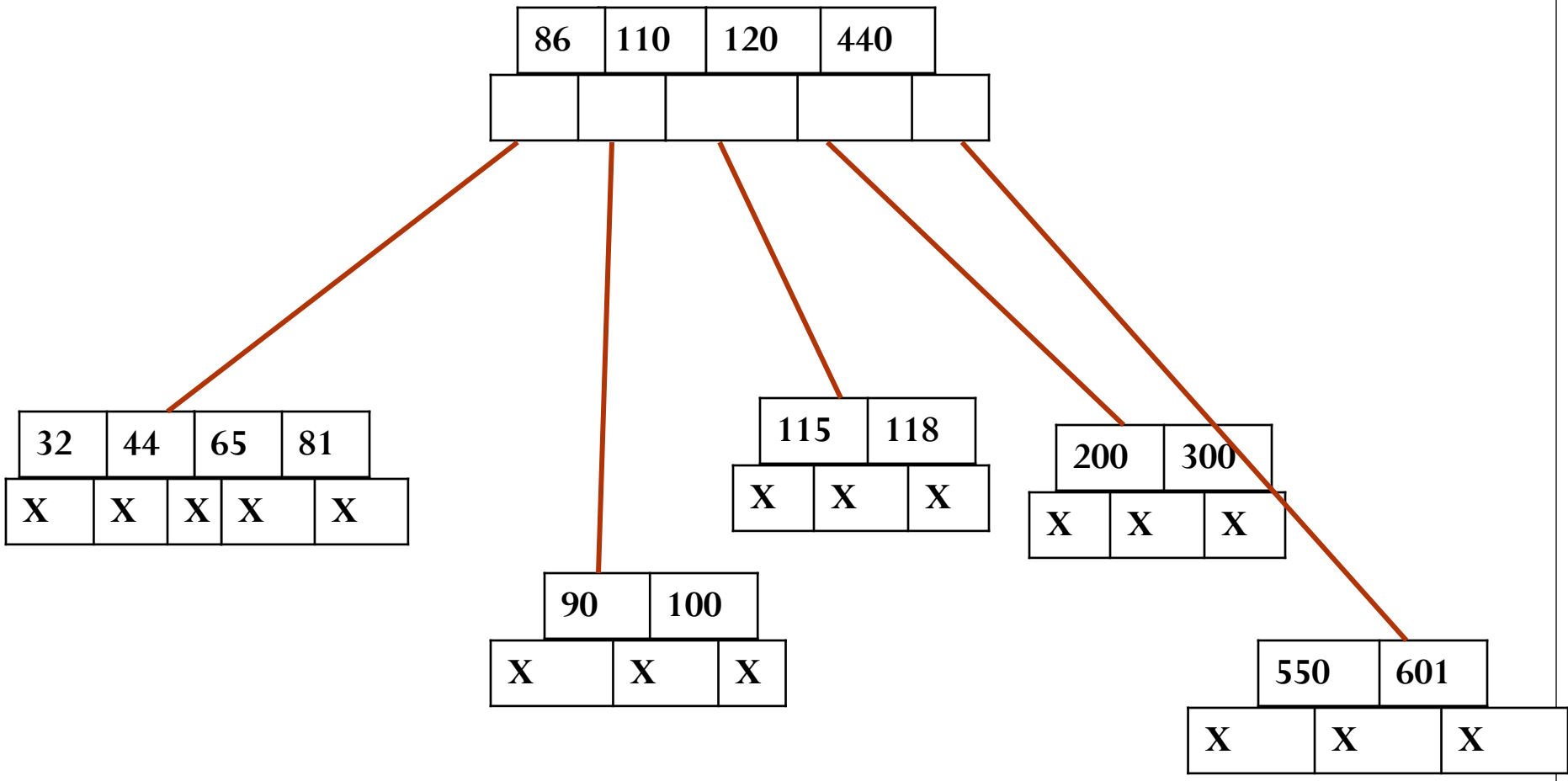


B-tree after deleting 95, 226, 221

B-tree of Order 5



B-tree of Order 5



B-tree after deleting 95, 226, 221, 70

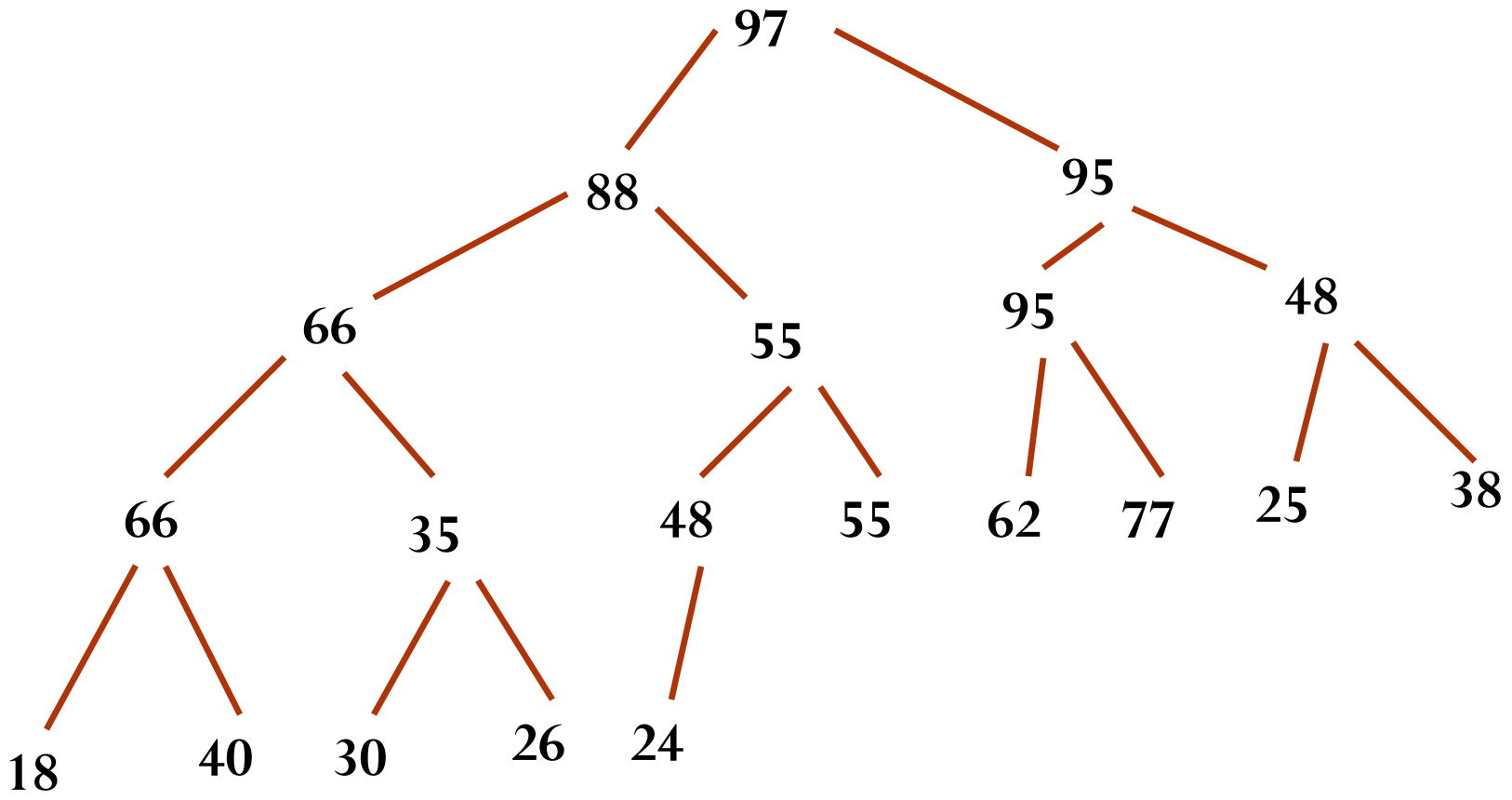
Heap

Suppose **H** is a complete binary tree with **n** elements

H is called a **heap or maxheap** if each node **N** of H has the following property

Value at **N** is greater than or equal to the value at each of the children of N.

Heap



97	88	95	66	55	95	48	66	35	48	55	62	77	25	38	18	40	30	26	24
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20

Inserting into a Heap

Suppose H is a heap with N elements

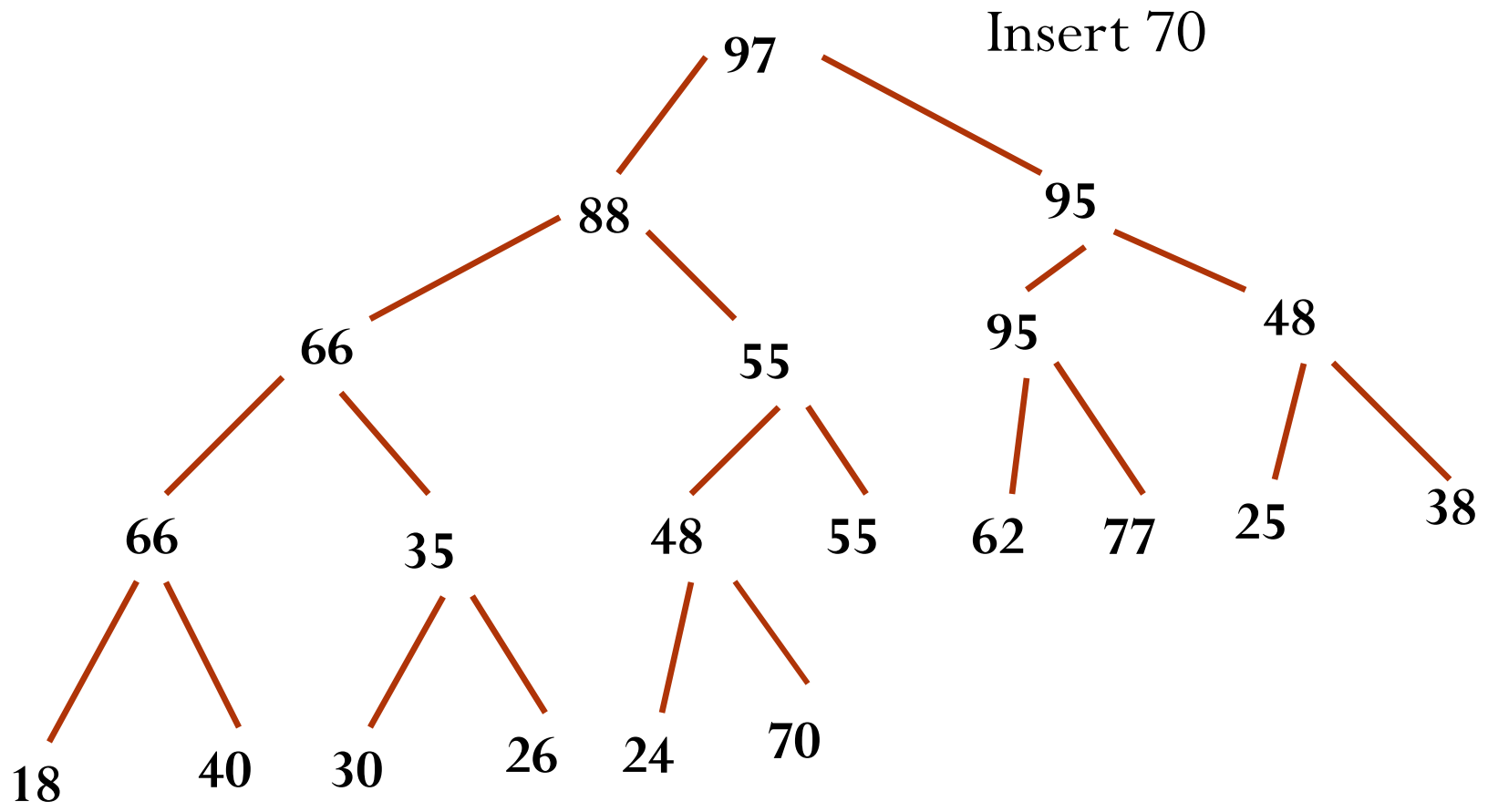
Suppose an ITEM of information is given.

Insertion of ITEM into heap H is given as follows:

[1] First adjoin ITEM at the end of H so that H is still a complete tree, but necessarily a heap

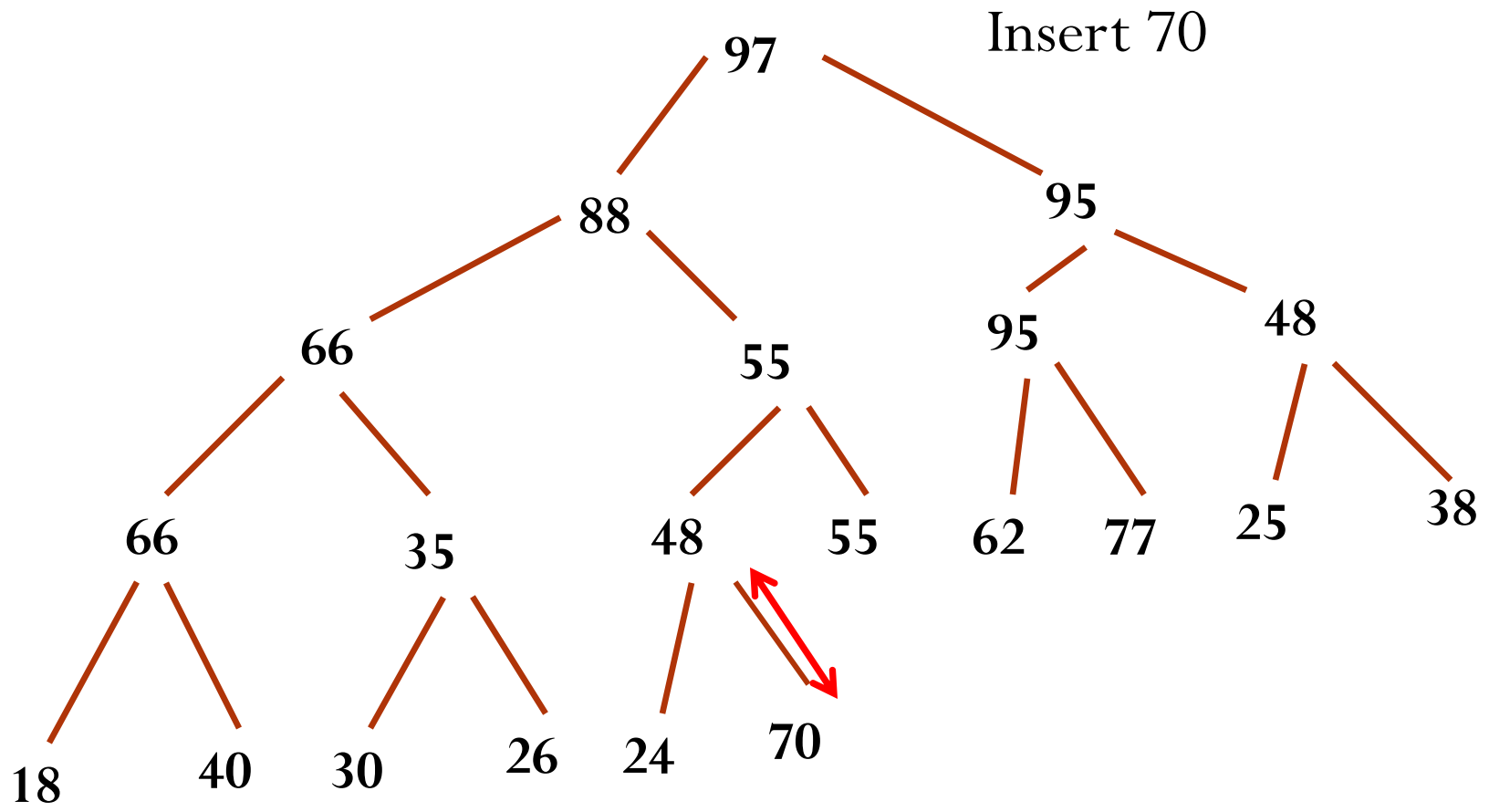
[2] Let ITEM rise to its appropriate place in H so that H is finally a heap

Heap



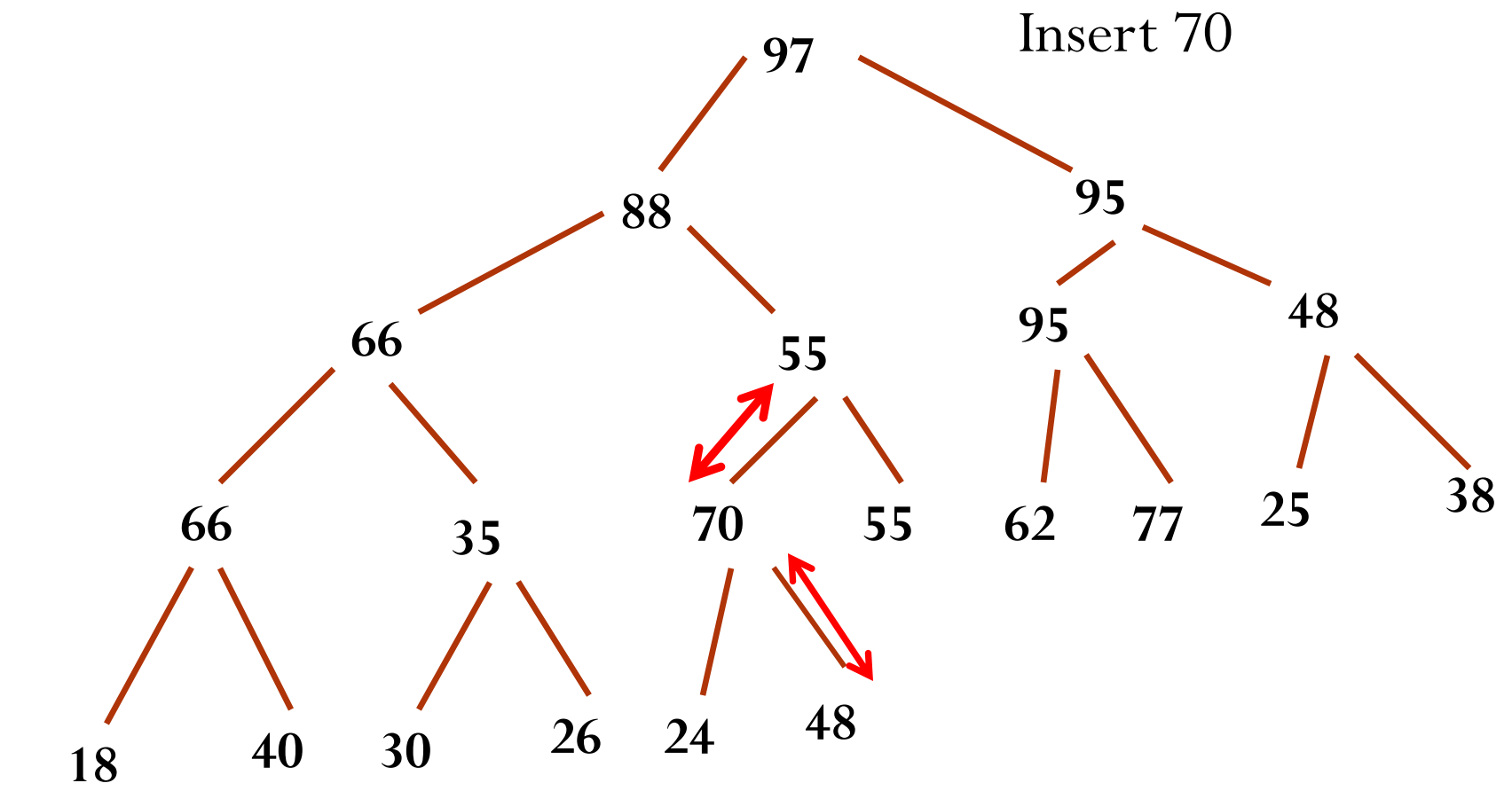
97	88	95	66	55	95	48	66	35	48	55	62	77	25	38	18	40	30	26	24
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20

Heap



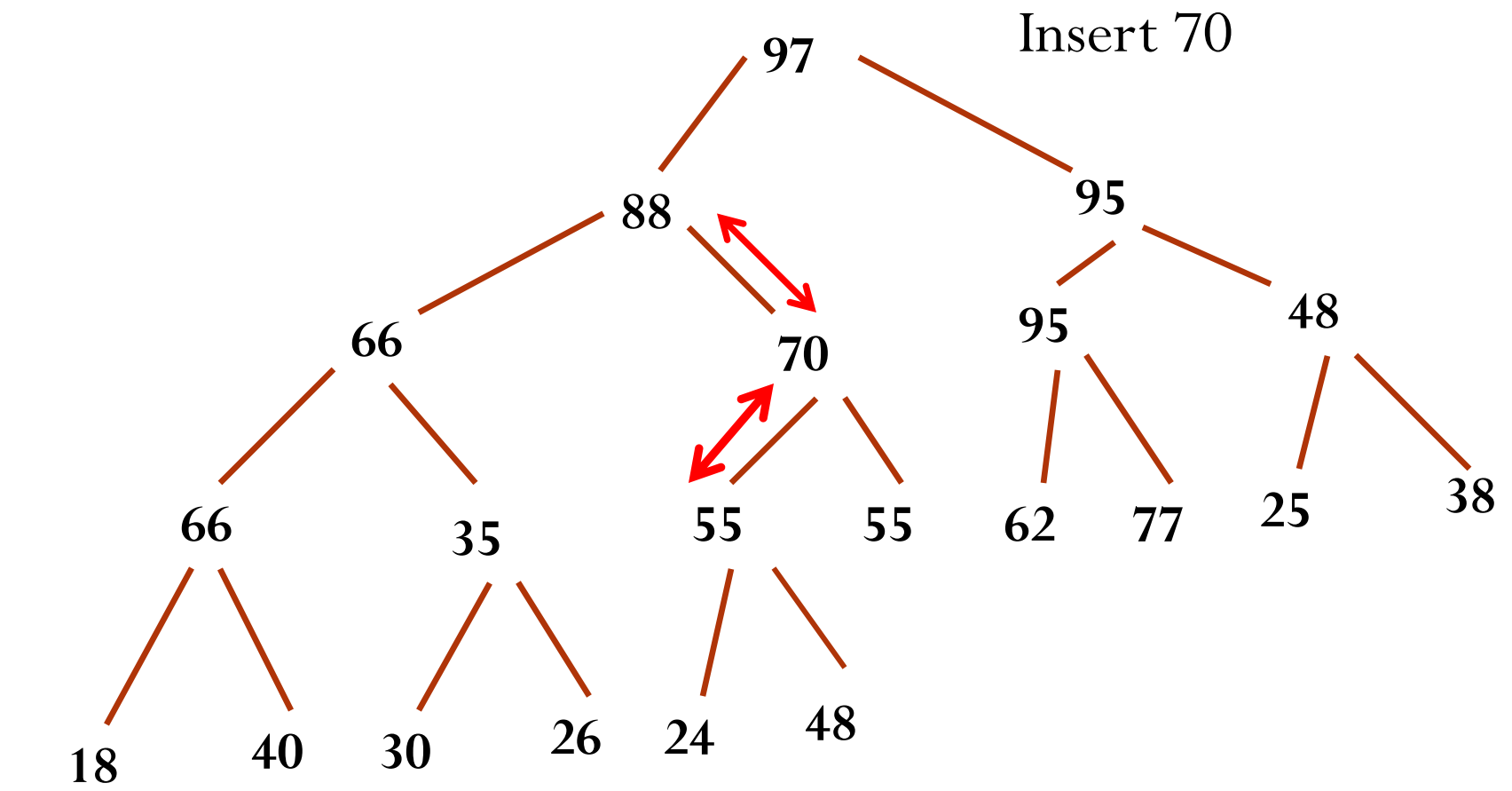
97	88	95	66	55	95	48	66	35	48	55	62	77	25	38	18	40	30	26	24
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20

Heap



97	88	95	66	55	95	48	66	35	48	55	62	77	25	38	18	40	30	26	24
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20

Heap



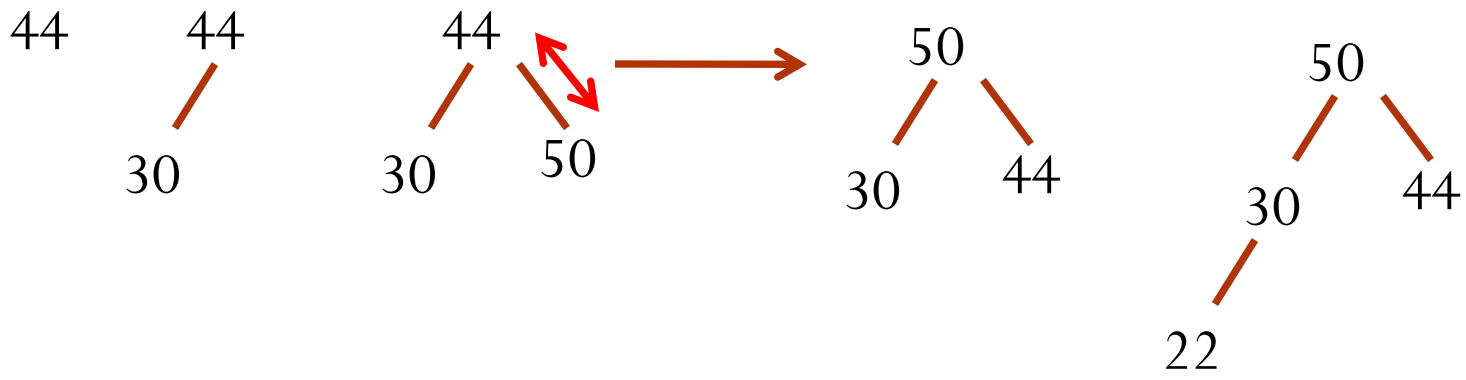
97	88	95	66	55	95	48	66	35	48	55	62	77	25	38	18	40	30	26	24
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20

Build a Heap

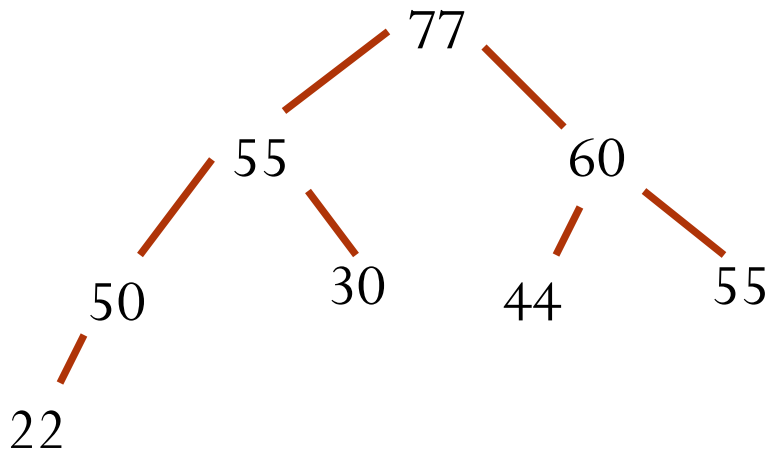
Build a heap from the following list

44, 30, 50, 22, 60, 55, 77, 55

44, 30, 50, 22, 60, 55, 77, 55



Complete the Rest Insertion



Deleting the Root of a Heap

Suppose H is a heap with N elements

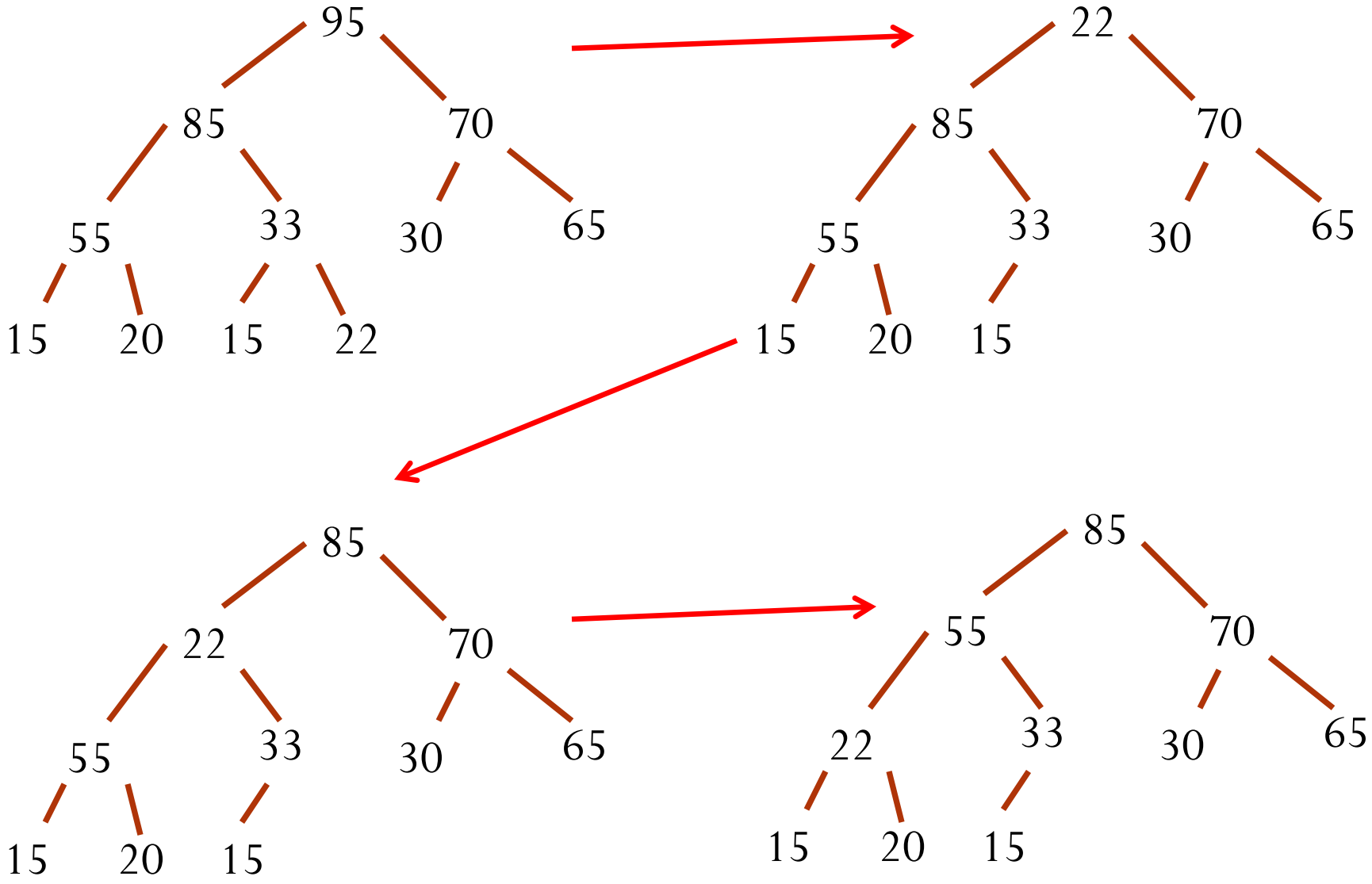
Suppose we want to delete the root R of H

Deletion of root is accomplished as follows

[1] Assign the root R to some variable ITEM

[2] Replace the deleted node R by the last node L of H so that H is still a complete tree but necessarily a heap

[3] Reheap. Let L sink to its appropriate place in H so that H is finally a heap.


95
85
70
55
33
30
65
15
20
15
22
22
85
70
55
33
30
65
15
20
15
85
22
70
55
33
30
65
15
20
15
85
55
70
22
33
30
65
15
20
15