Digital Search Trees & Binary Tries

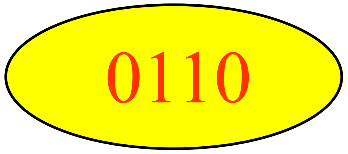
- **Analog of radix sort to searching.**
- **Keys are binary bit strings.**
  - **Fixed length – 0110, 0010, 1010, 1011.**
  - **Variable length – 01, 00, 101, 1011.**
- **Application – IP routing, packet classification, firewalls.**
  - **IPv4 – 32 bit IP address.**
  - **IPv6 – 128 bit IP address.**

Digital Search Tree

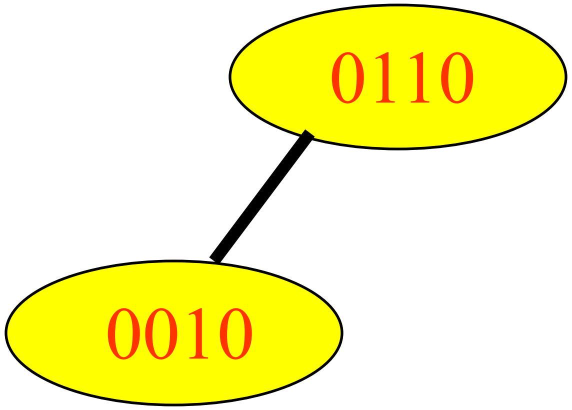
- Assume fixed number of bits.
- Not empty
  - Root contains one dictionary pair (any pair).
  - All remaining pairs whose key begins with a 0 are in the left subtree.
  - All remaining pairs whose key begins with a 1 are in the right subtree.
  - Left and right subtrees are digital subtrees on remaining bits.

Example

- **Start with an empty digital search tree and insert a pair whose key is 0110.**

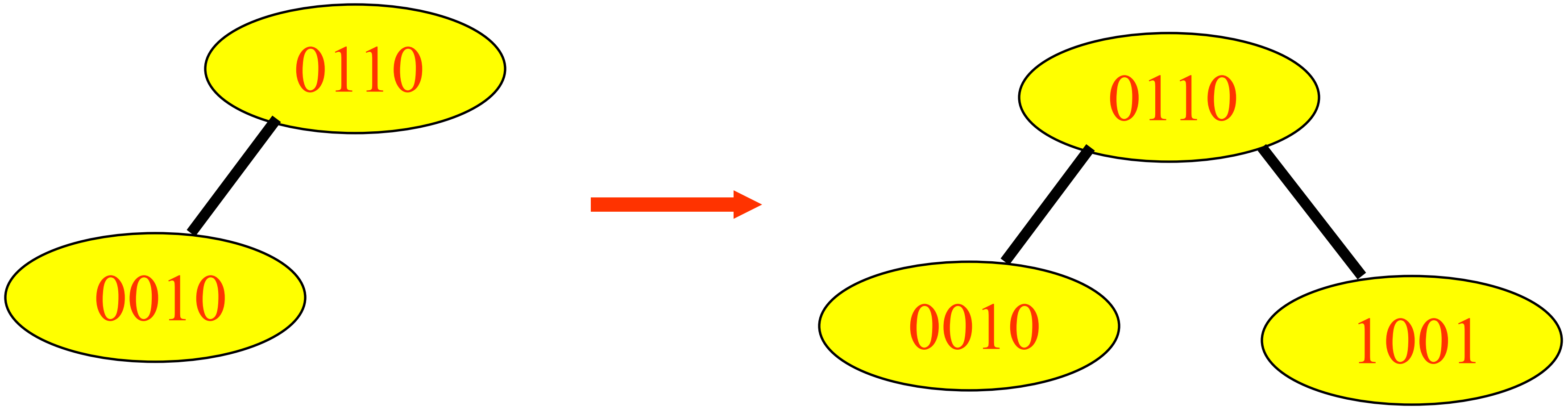


- Now, insert a pair whose key is 0010.



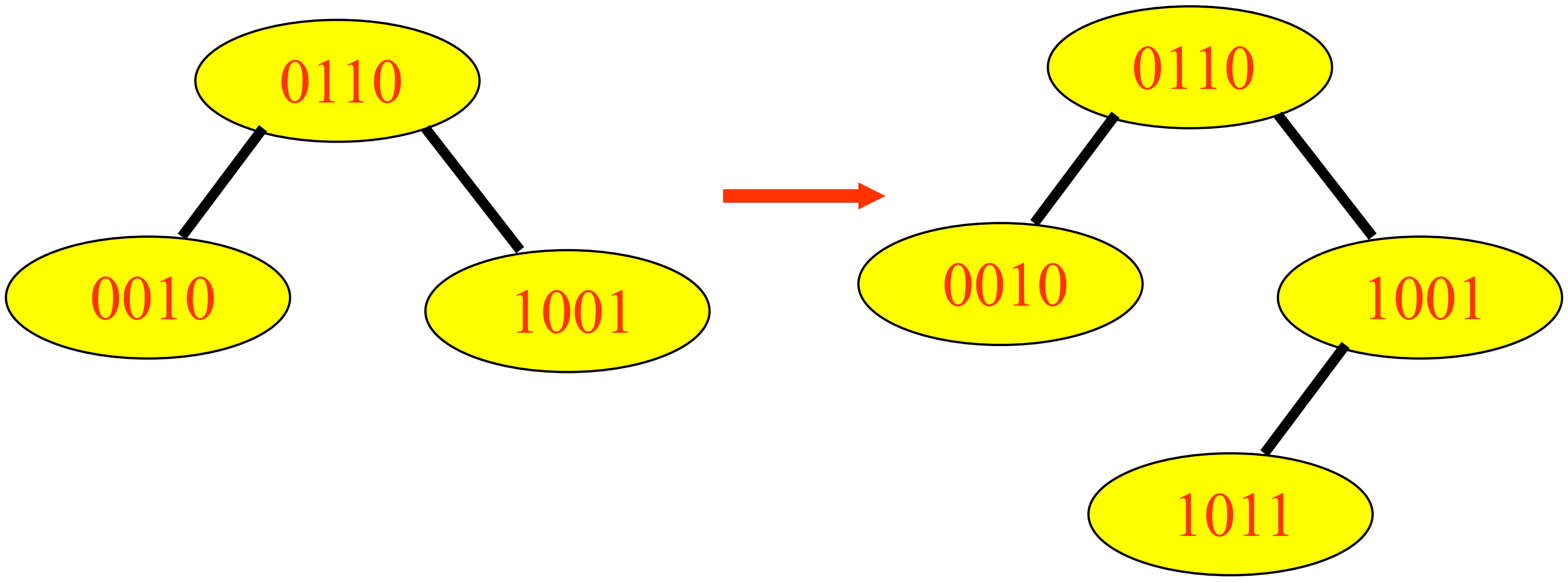
Example

- Now, insert a pair whose key is 1001.



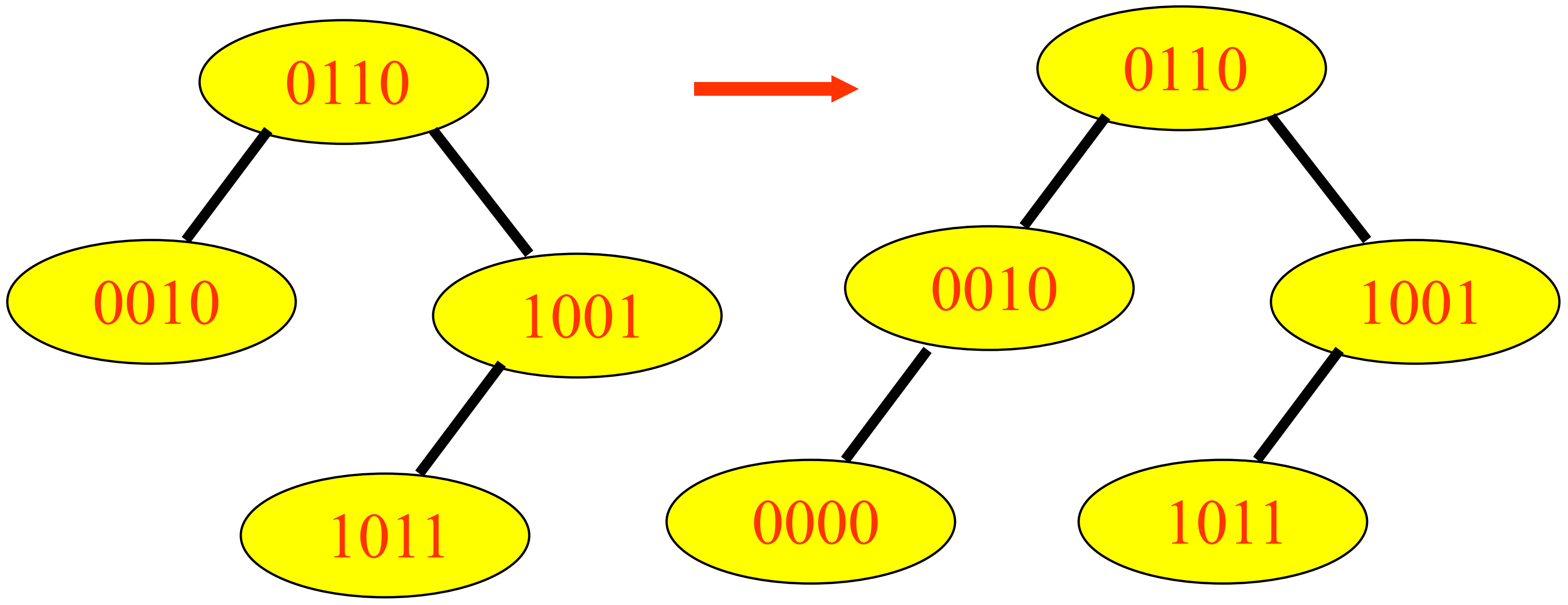
Example

- Now, insert a pair whose key is 1011.

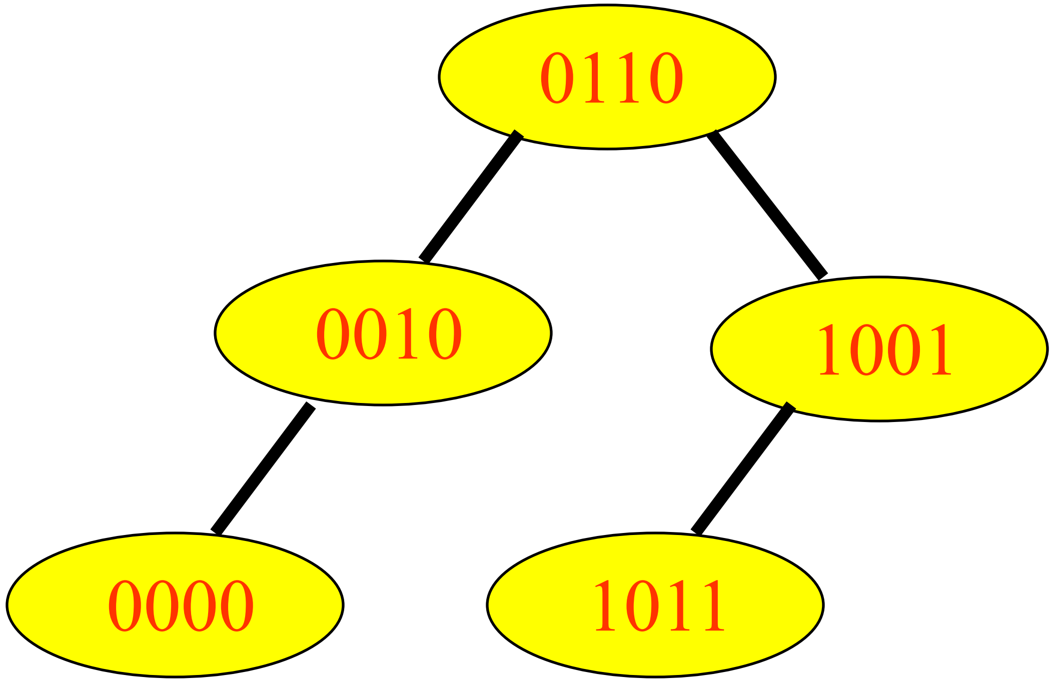


Example

- Now, insert a pair whose key is 0000.



Search/Insert/Delete

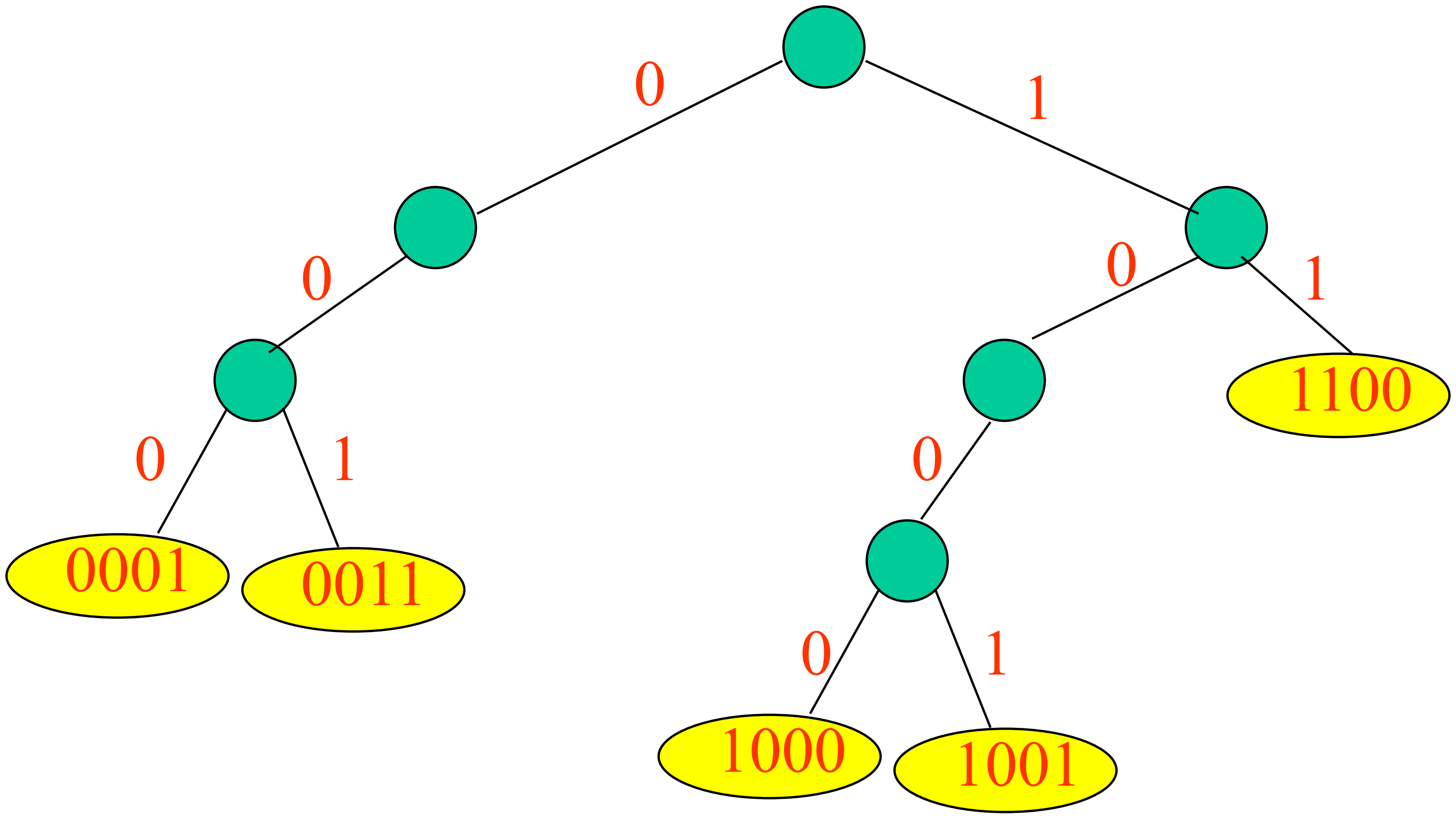


- Complexity of each operation is O(#bits in a key).
- #key comparisons = O(height).
- Expensive when keys are very long.

Binary Trie

- Information Retrieval.
- At most one key comparison per operation.
- Fixed length keys.
  - Branch nodes.
    - Left and right child pointers.
    - No data field(s).
  - Element nodes.
    - No child pointers.
    - Data field to hold dictionary pair.

Example

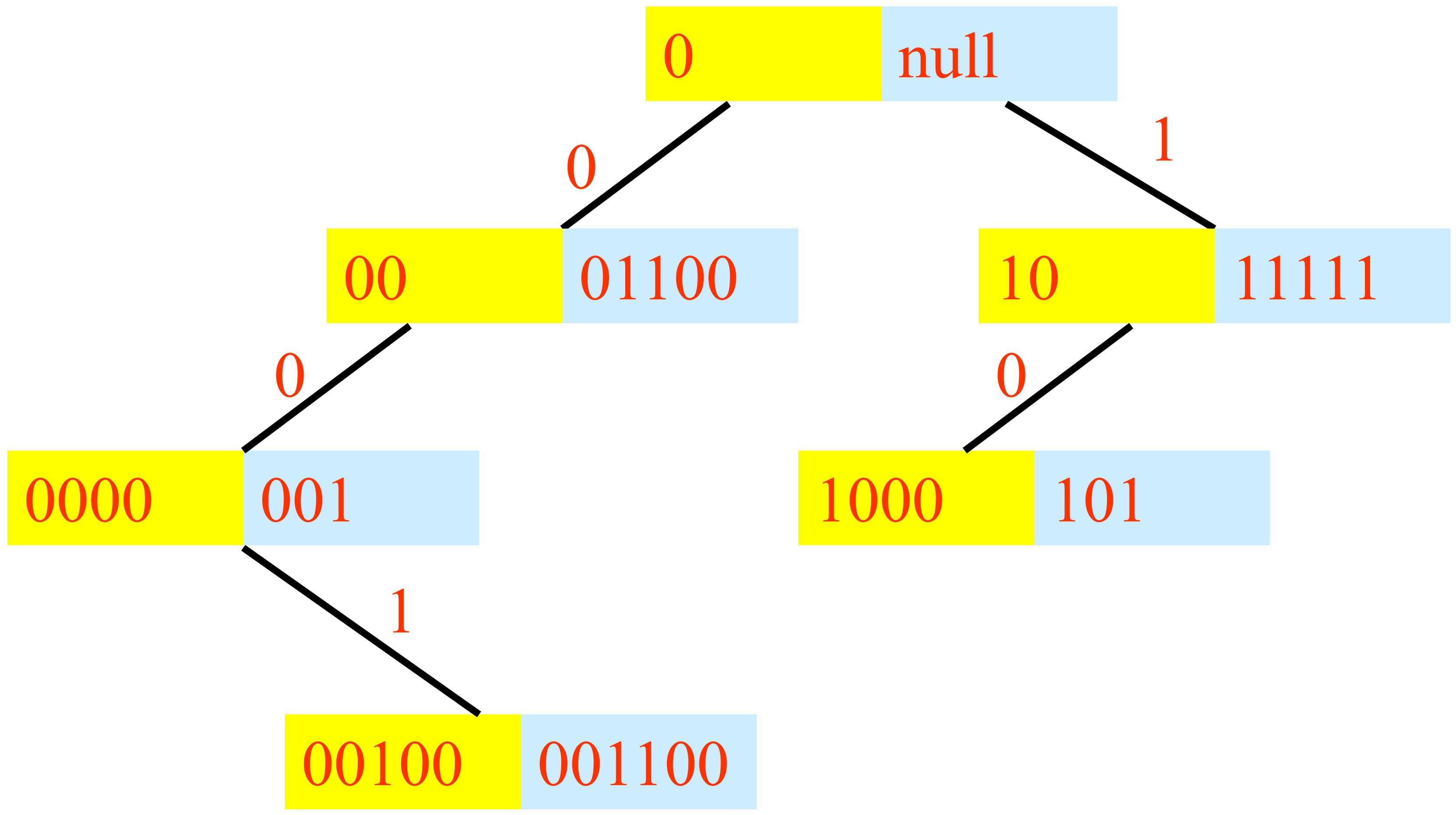


At most one key comparison for a search.

Variable Key Length

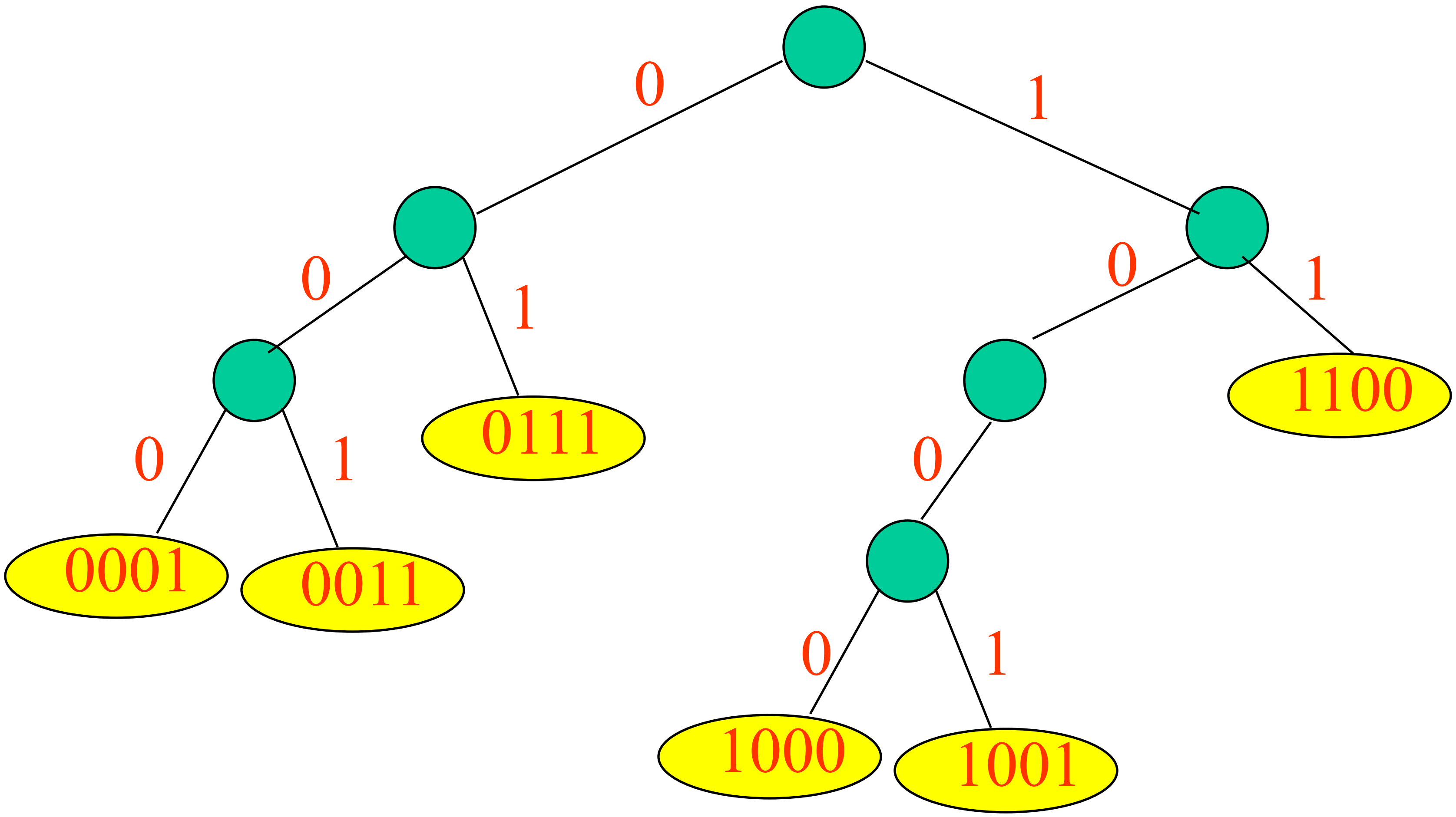
- Left and right child fields.
- Left and right pair fields.
  - Left pair is pair whose key terminates at root of left subtree or the single pair that might otherwise be in the left subtree.
  - Right pair is pair whose key terminates at root of right subtree or the single pair that might otherwise be in the right subtree.
  - Field is null otherwise.

Example



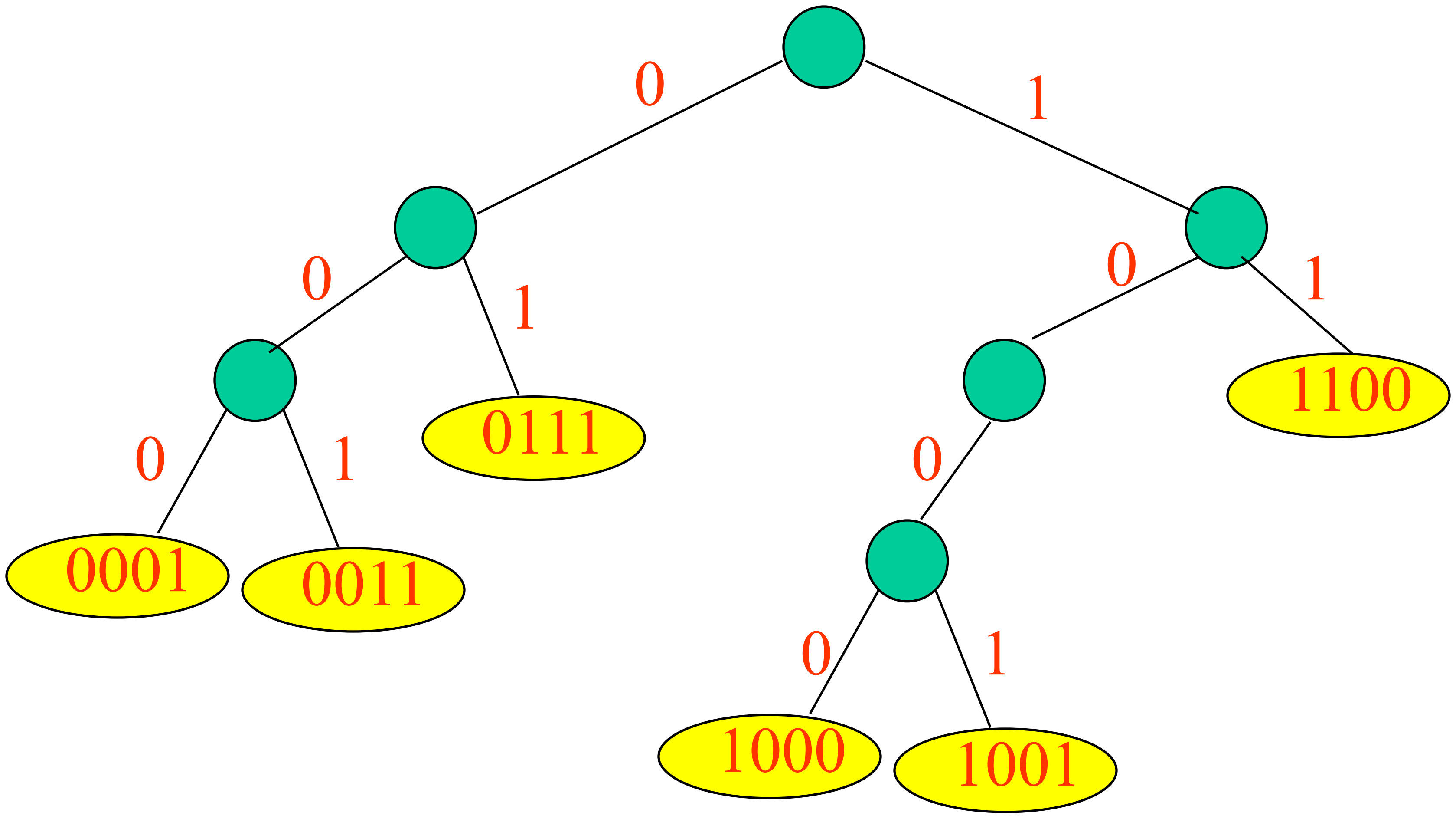
At most one key comparison for a search.

Fixed Length Insert



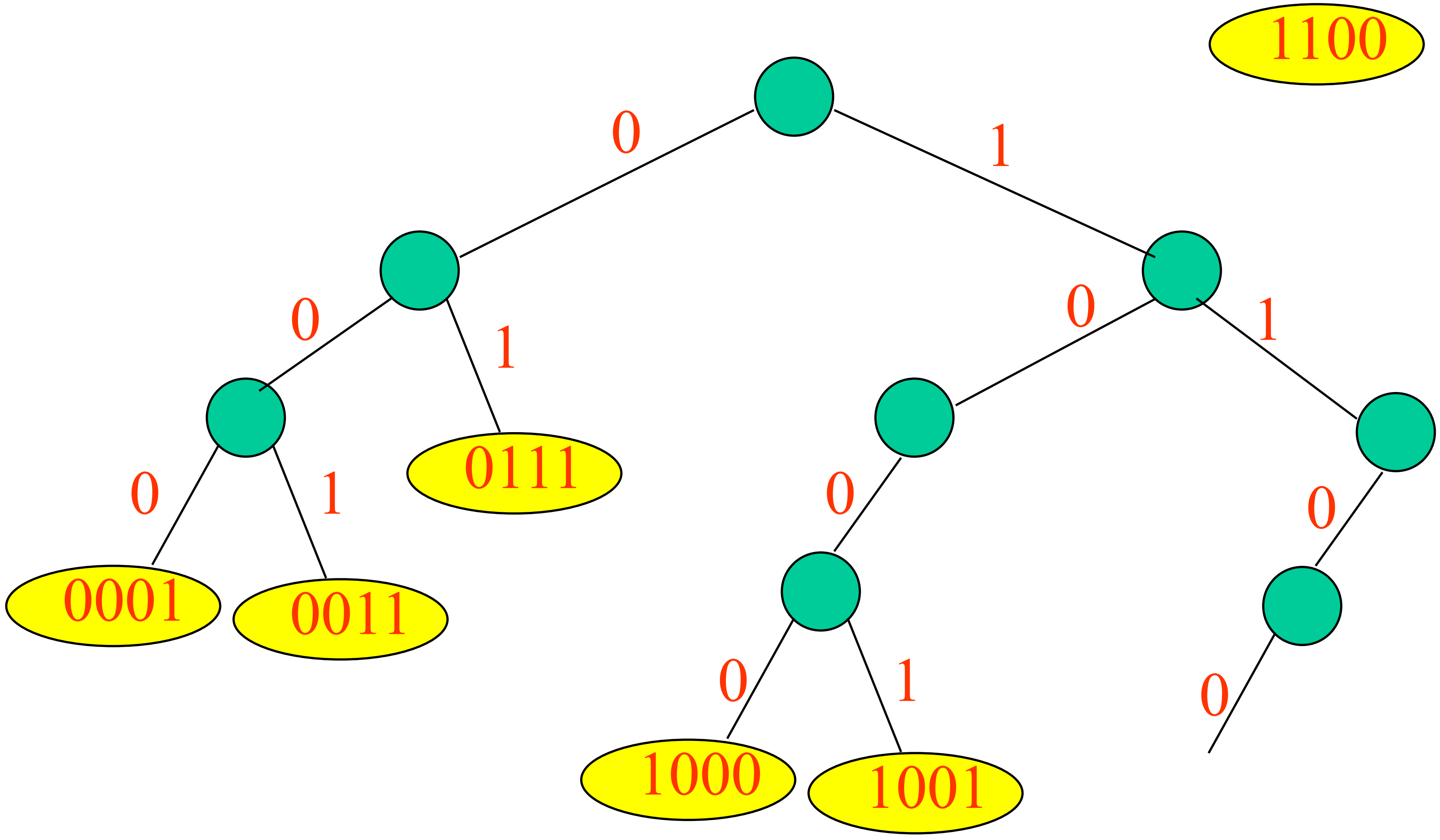
Insert 0111. Zero compares.

Fixed Length Insert



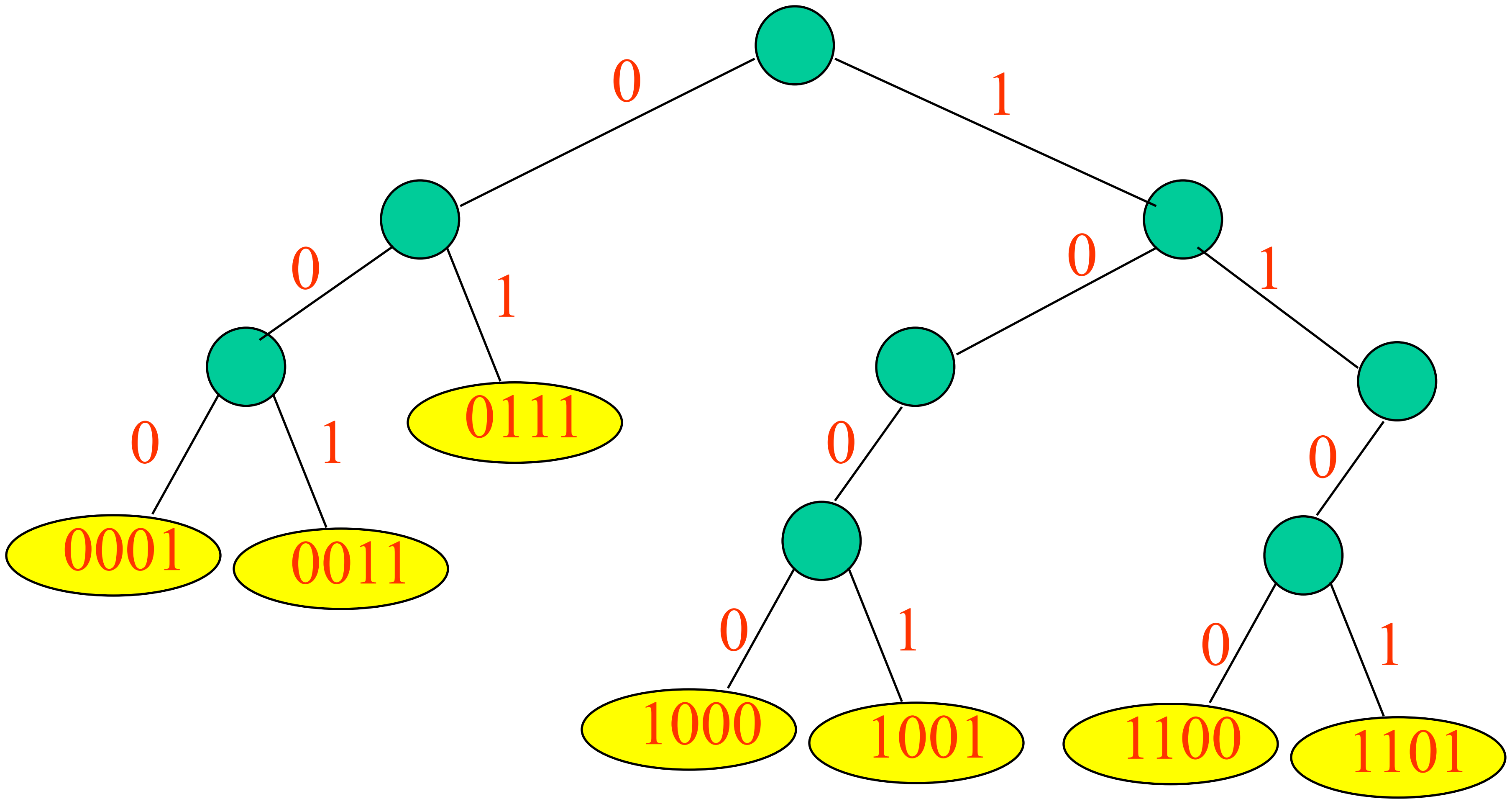
Insert 1101.

Fixed Length Insert



Insert 1101.

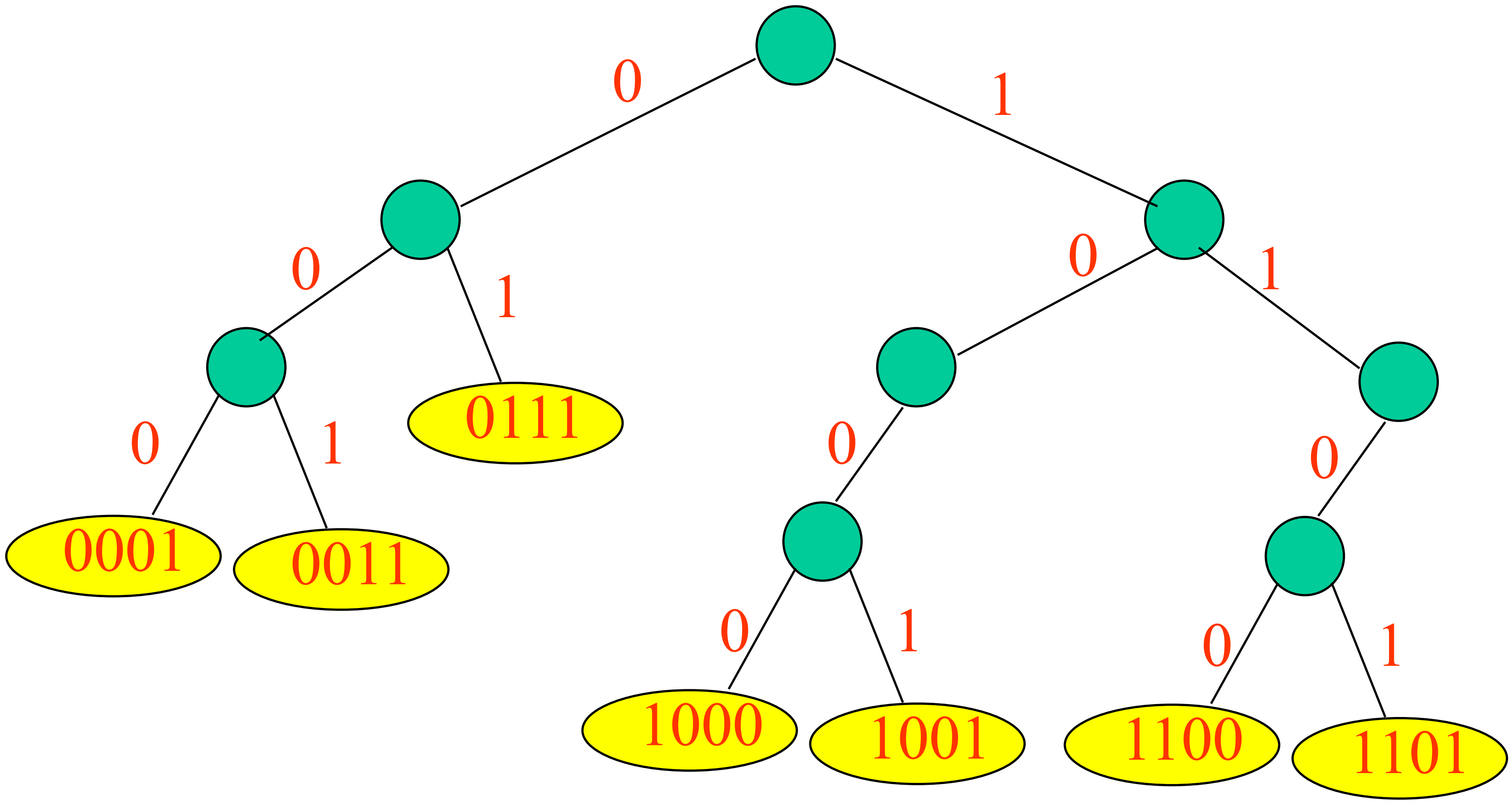
Fixed Length Insert



Insert 1101.

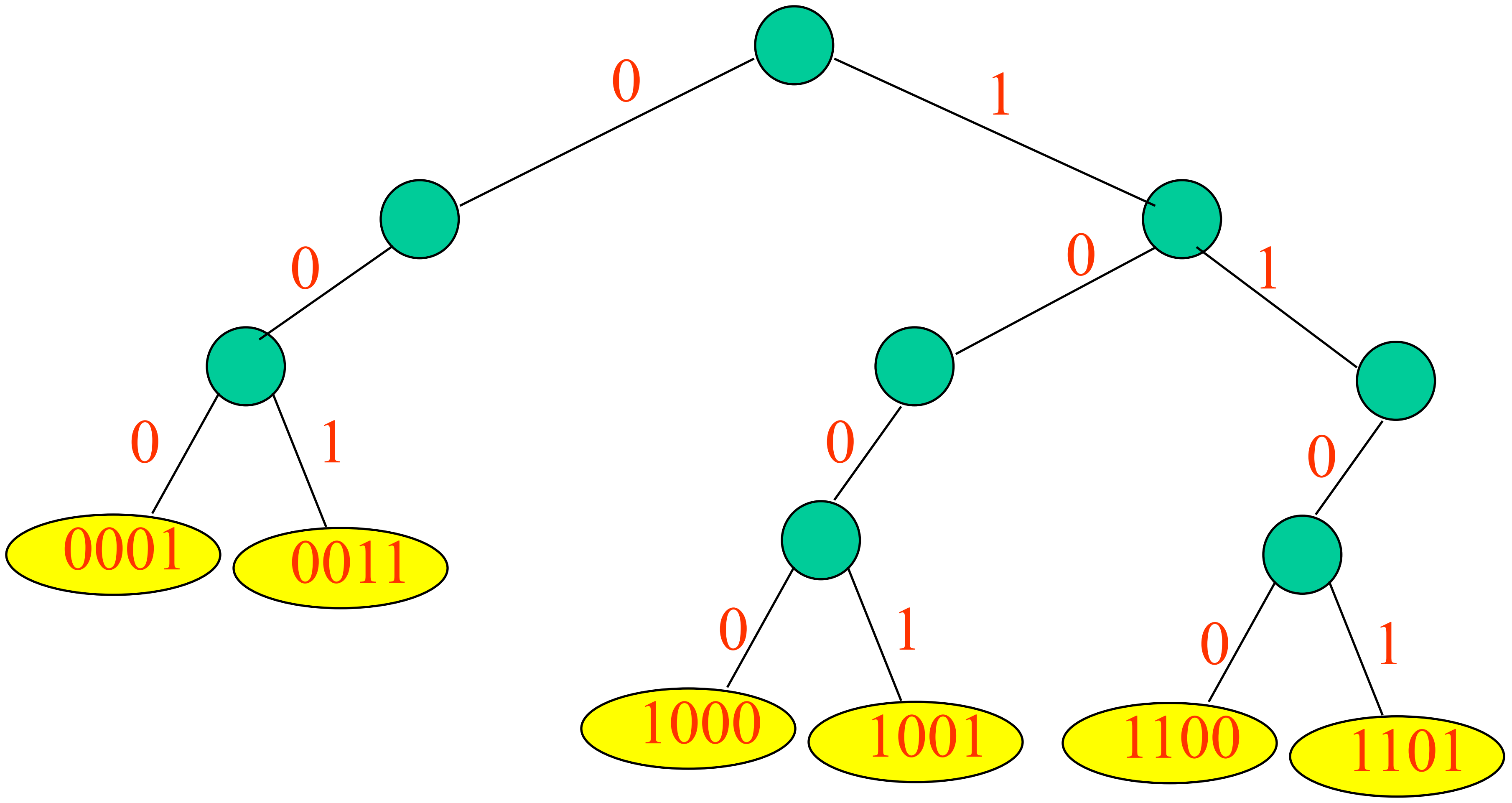
One compare.

Fixed Length Delete



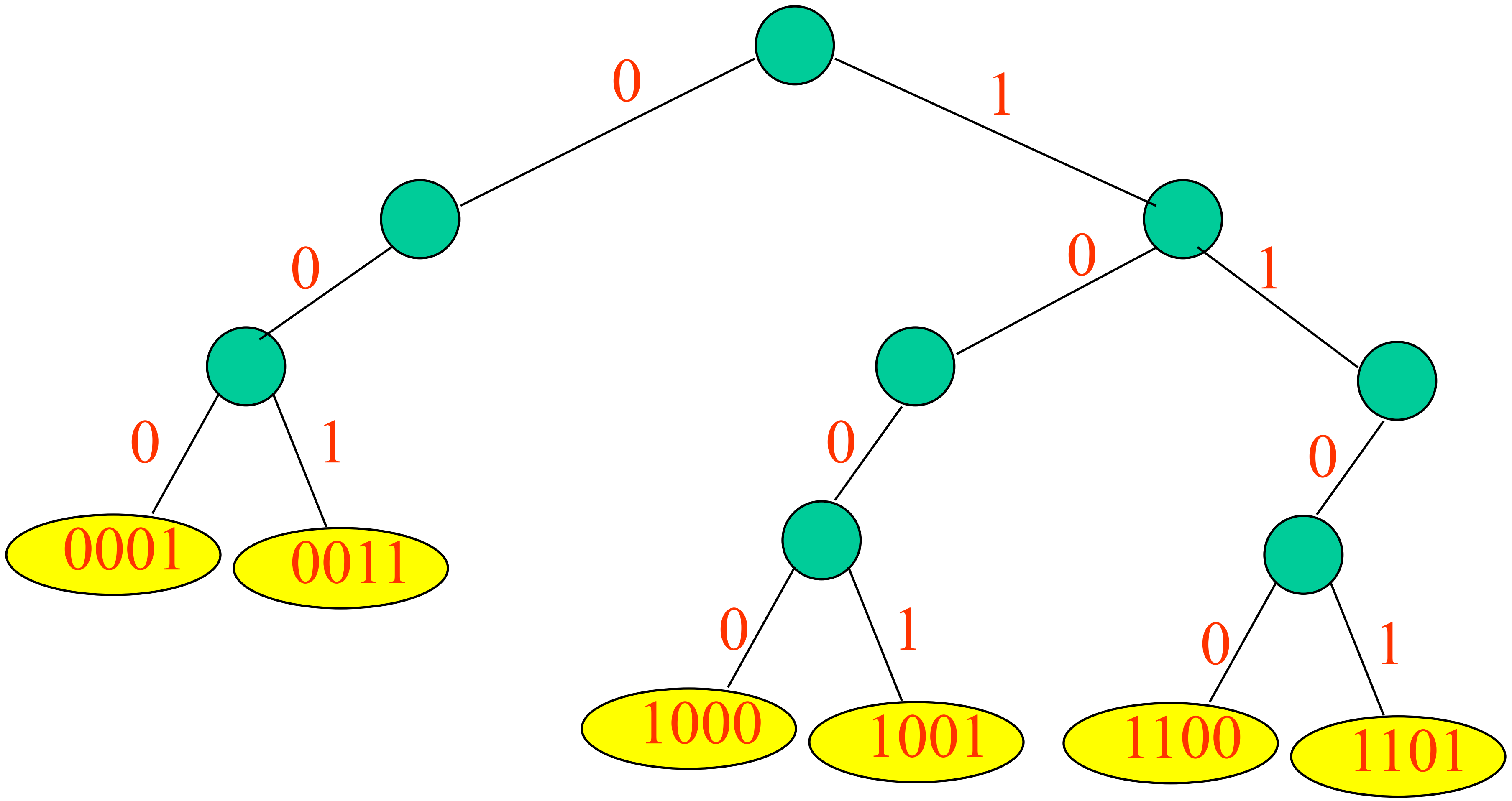
Delete 0111.

Fixed Length Delete



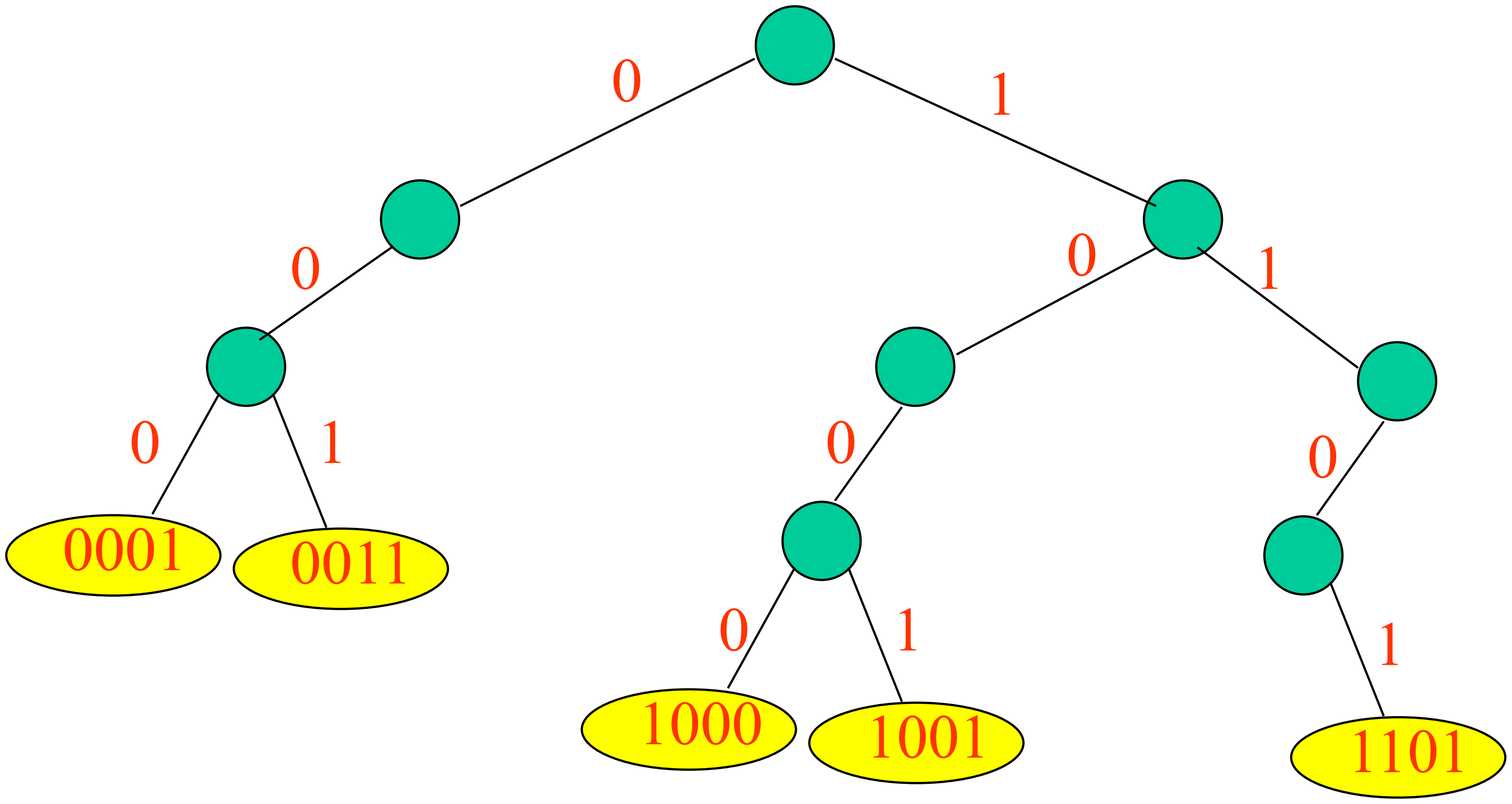
Delete 0111. One compare.

Fixed Length Delete



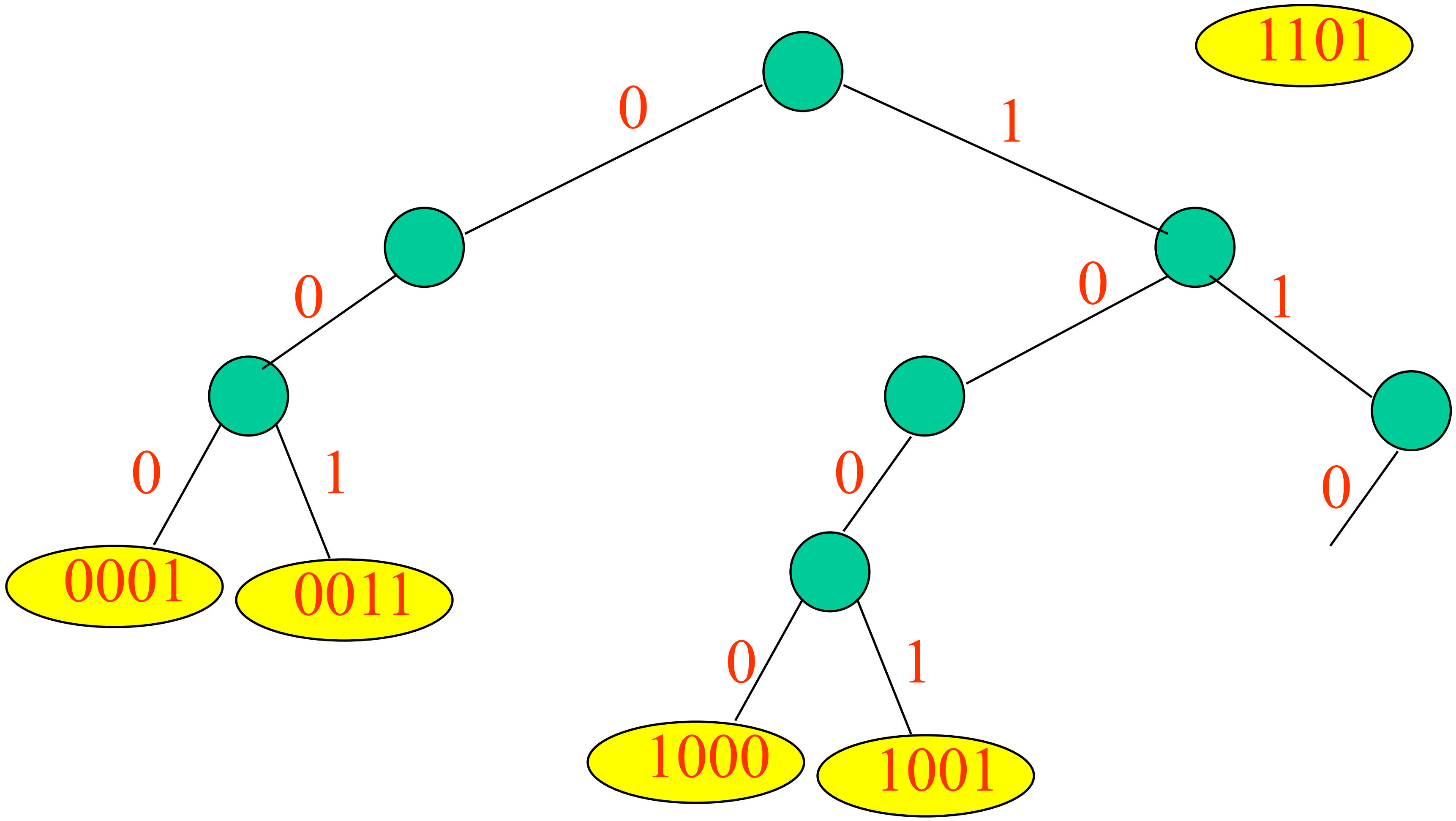
Delete 1100.

Fixed Length Delete



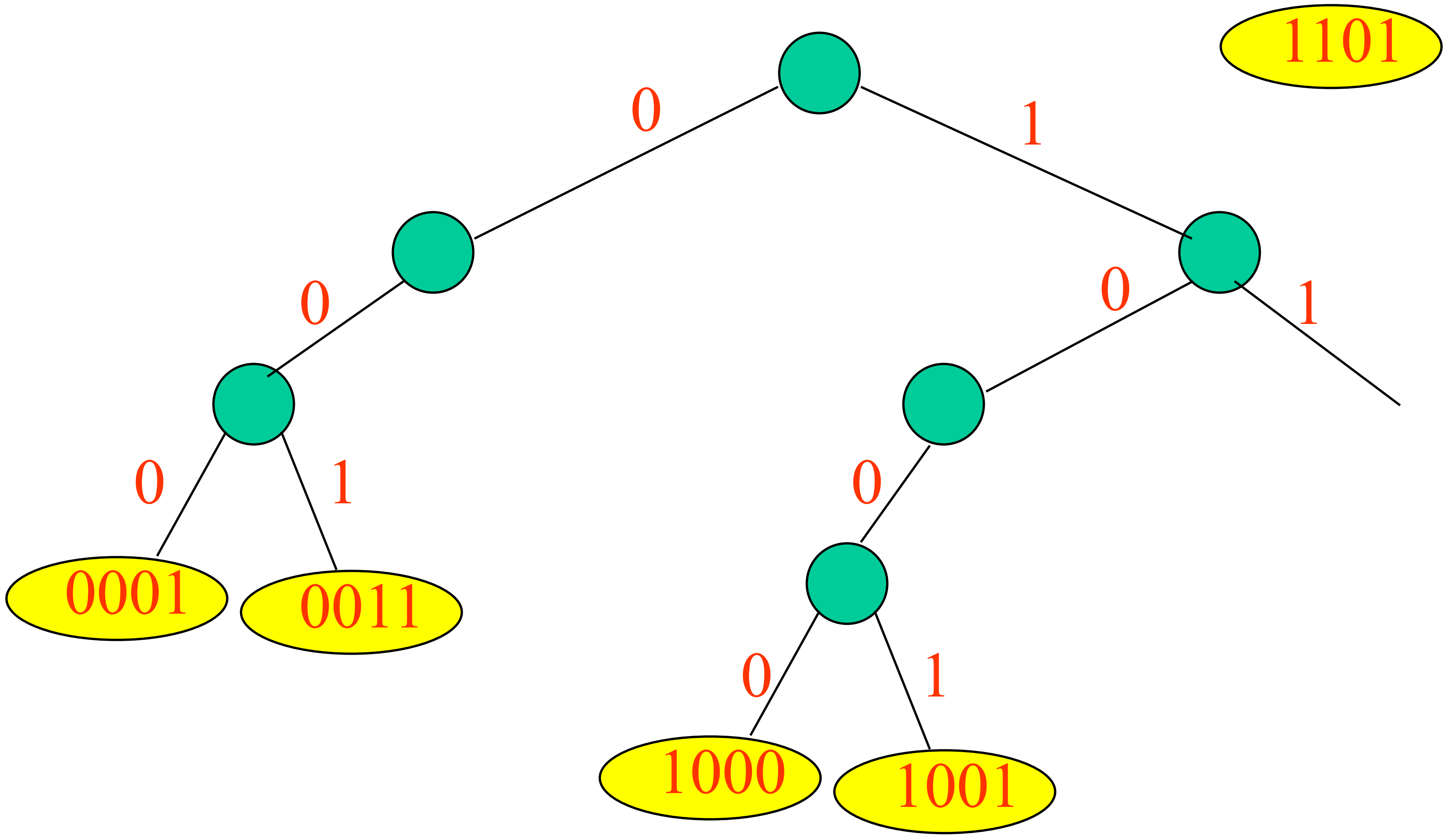
Delete 1100.

Fixed Length Delete



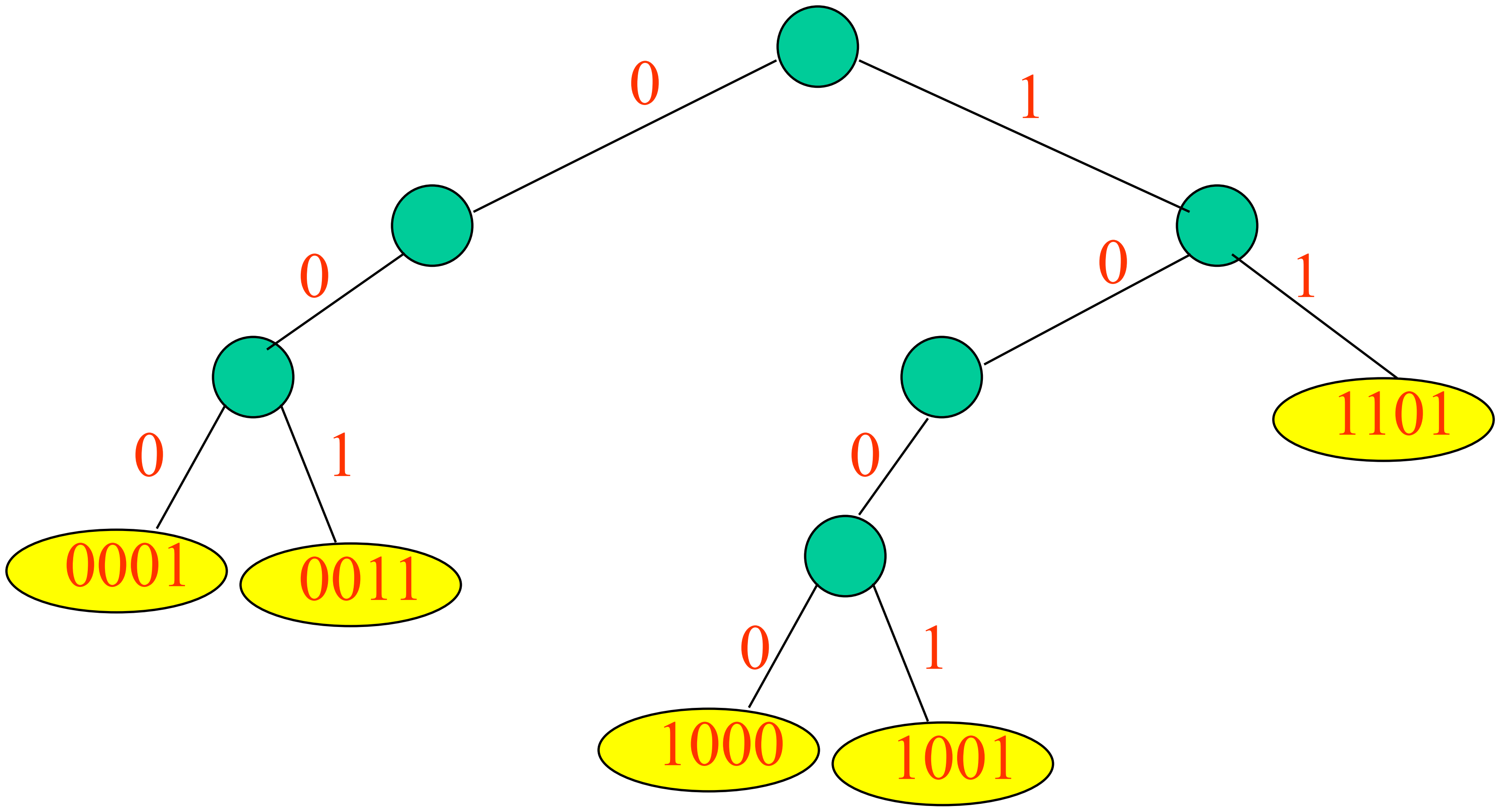
Delete 1100.

Fixed Length Delete



Delete 1100.

Fixed Length Delete

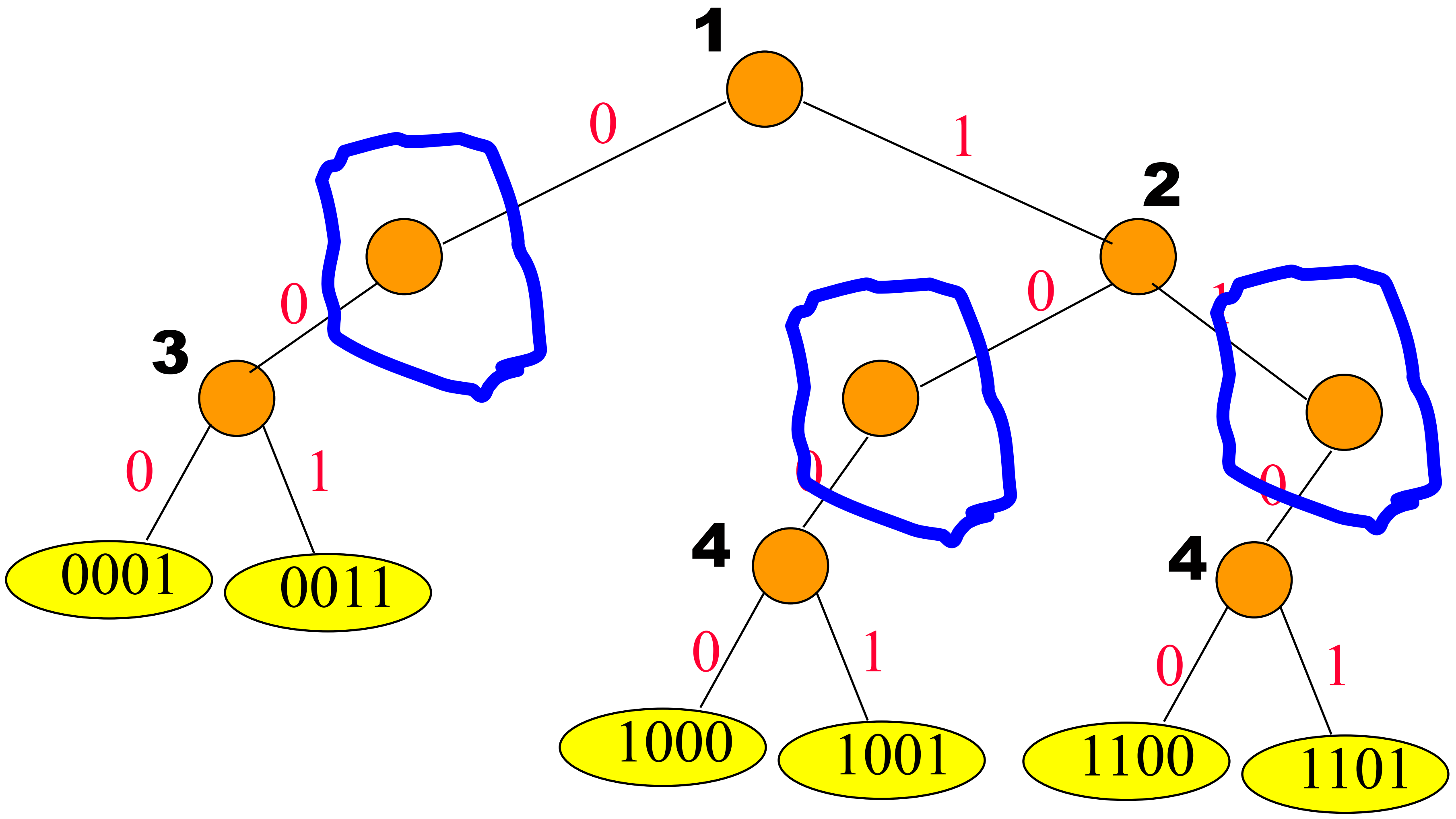


Delete 1100. One compare.

Compressed Binary Tries

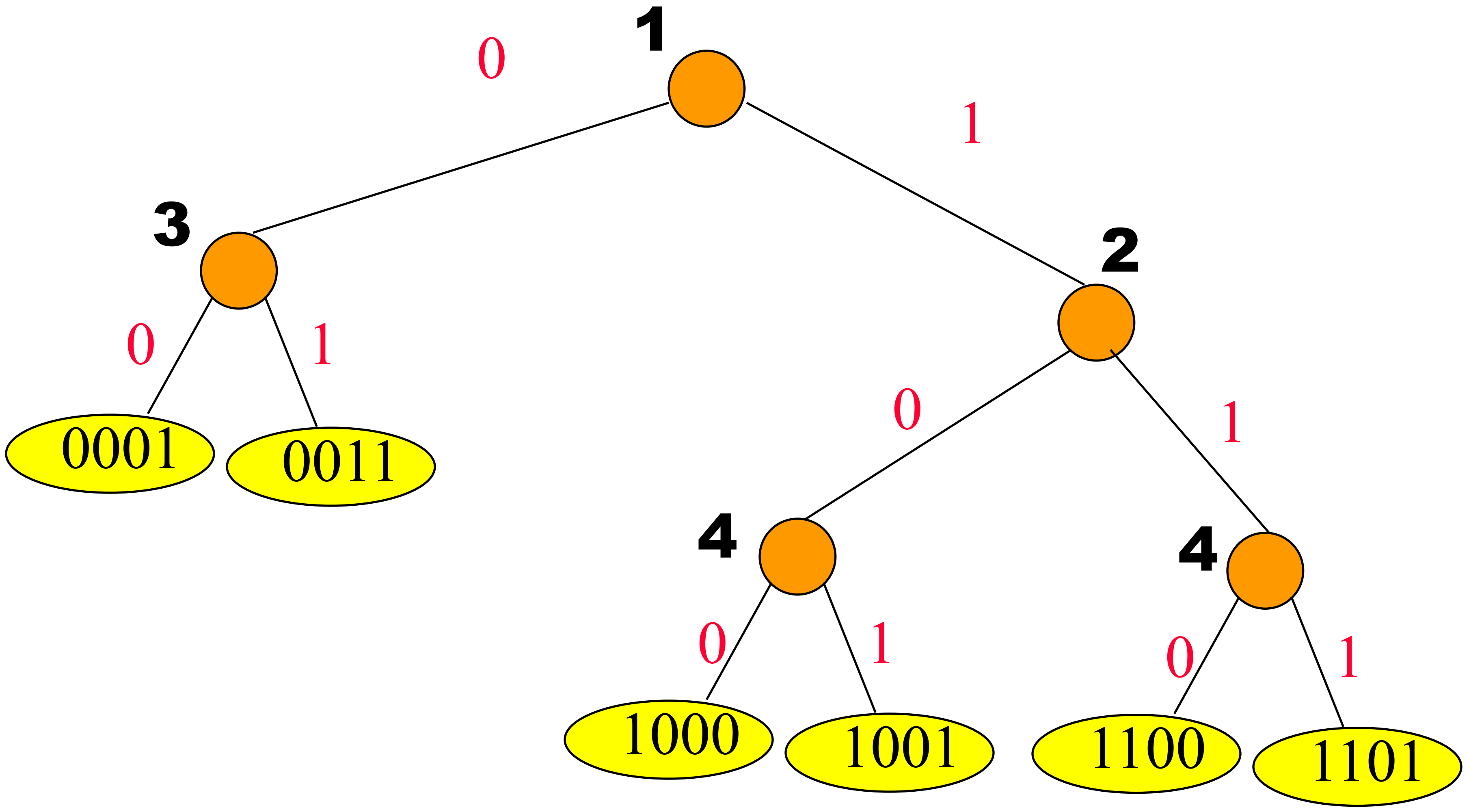
- No branch node whose degree is 1.
- Add a bit# field to each branch node.
- bit# tells you which bit of the key to use to decide whether to move to the left or right subtrie.

Binary Trie



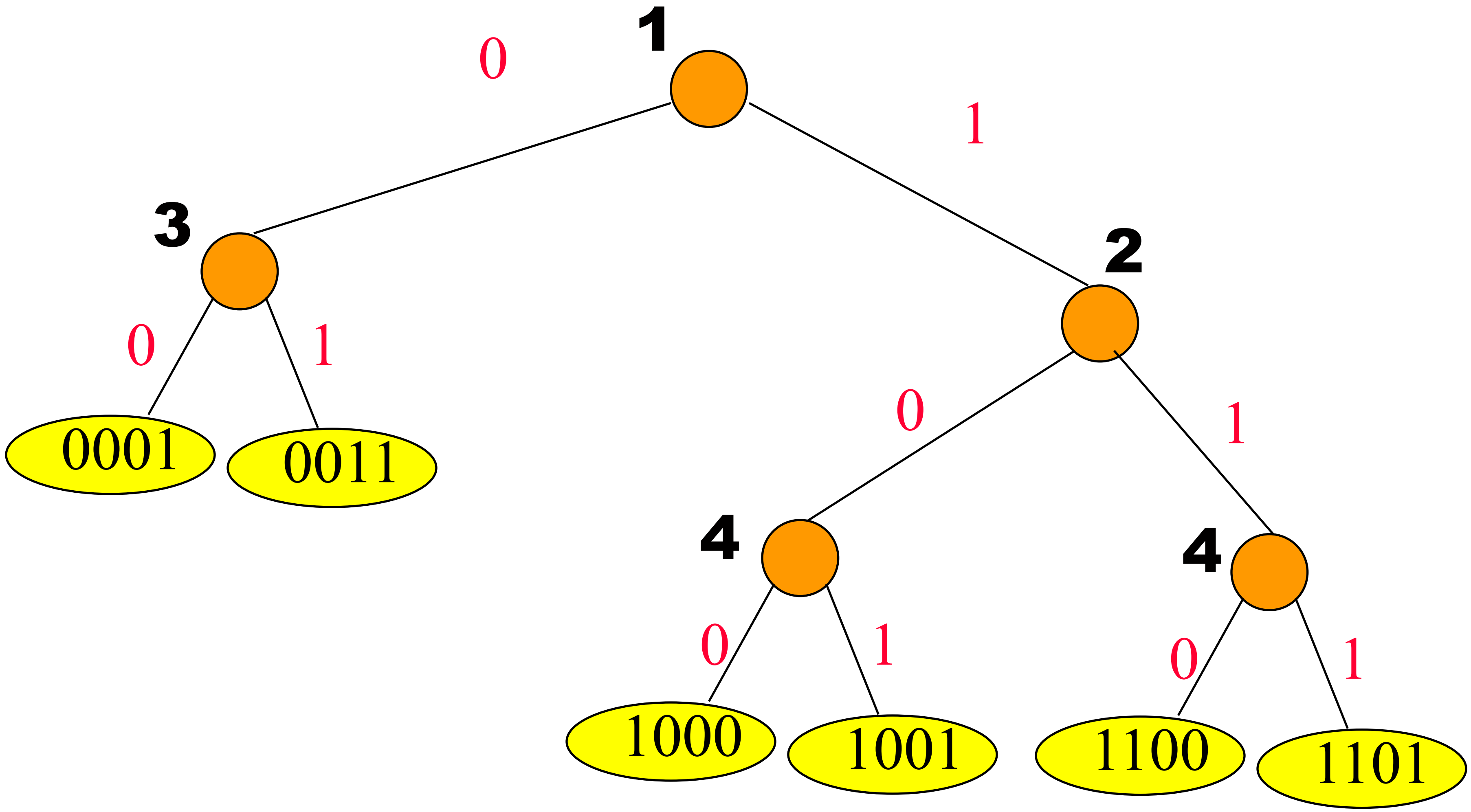
bit# field shown in black outside branch node.

Compressed Binary Trie



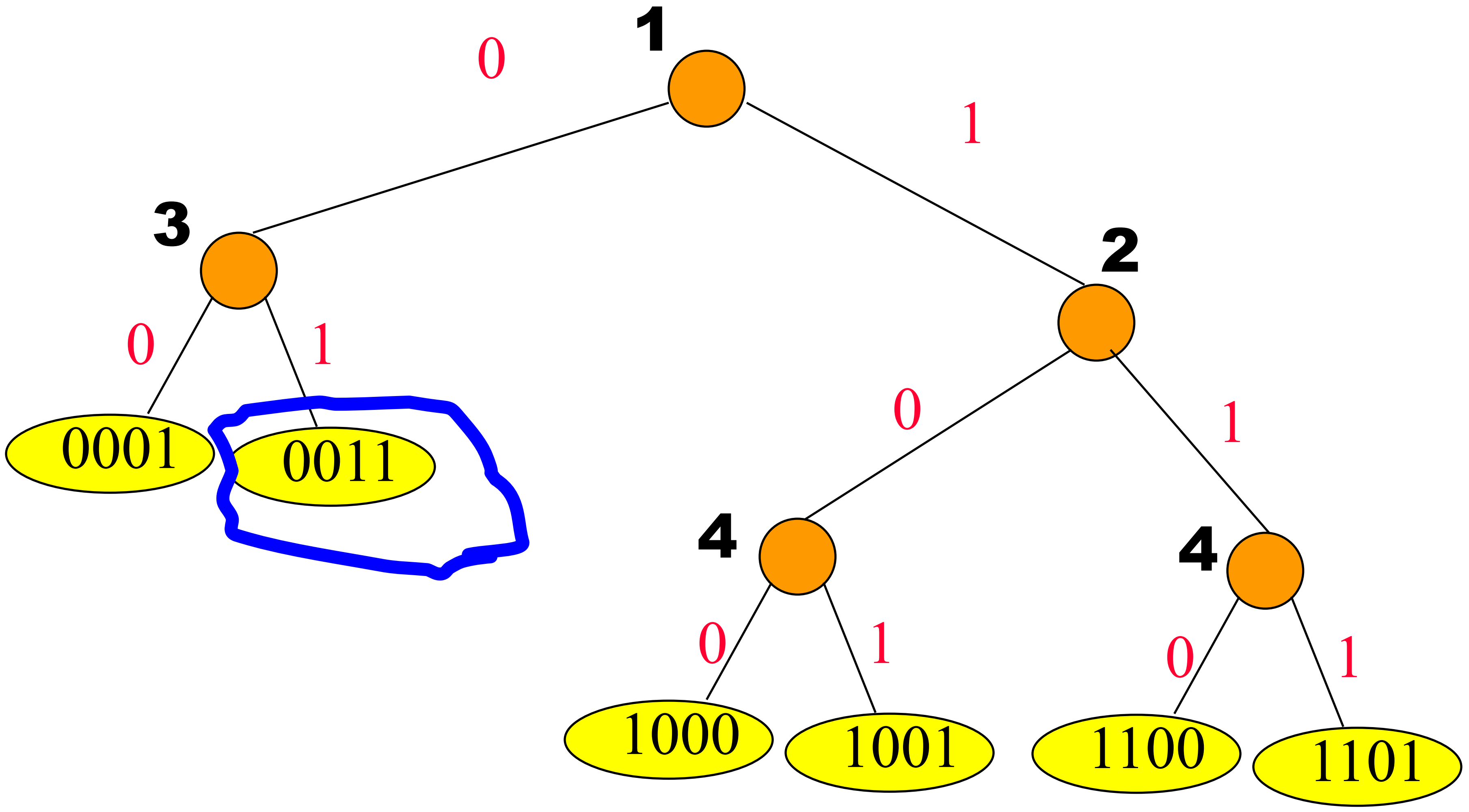
bit# field shown in black outside branch node.

Compressed Binary Trie



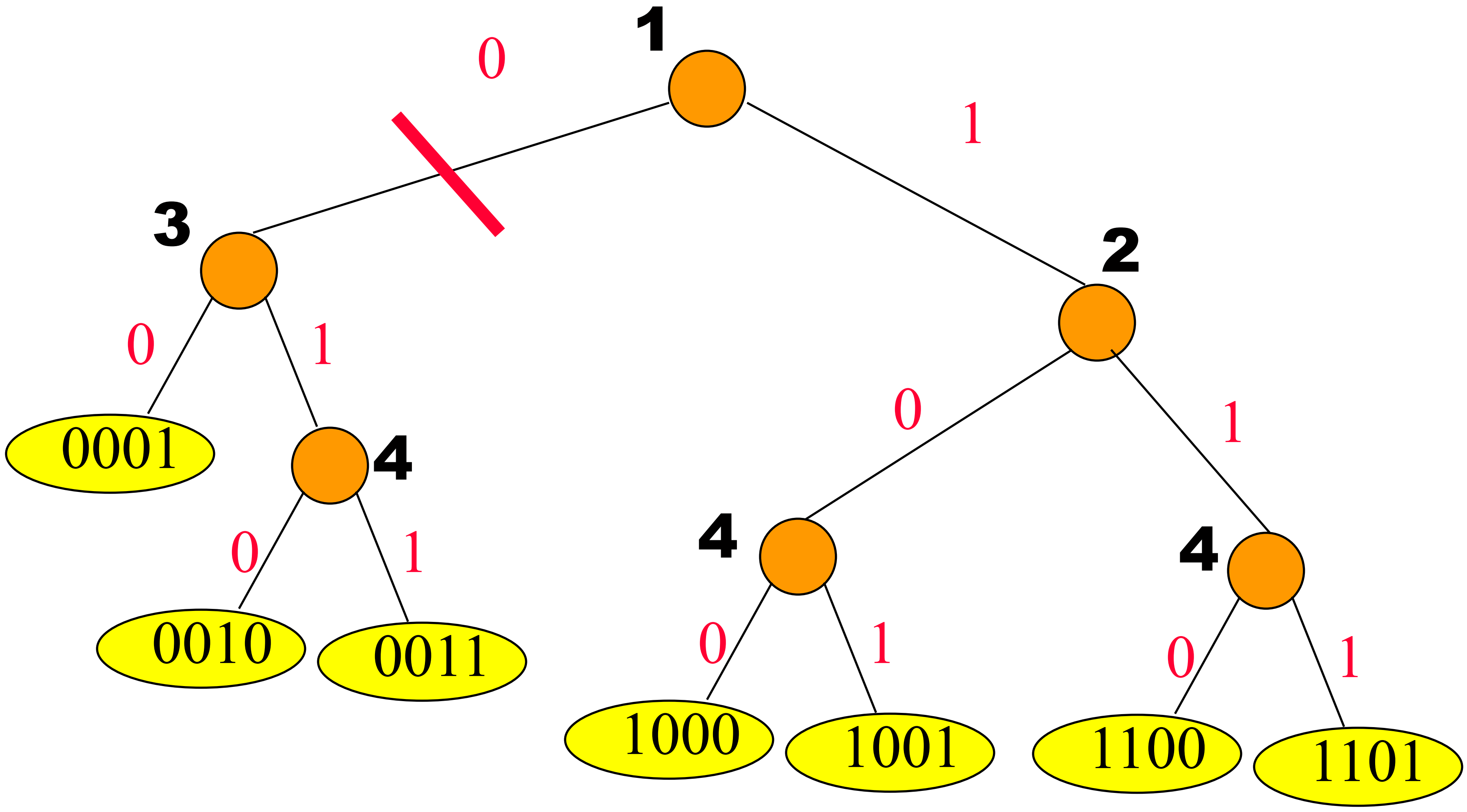
#branch nodes = n – 1.

Insert



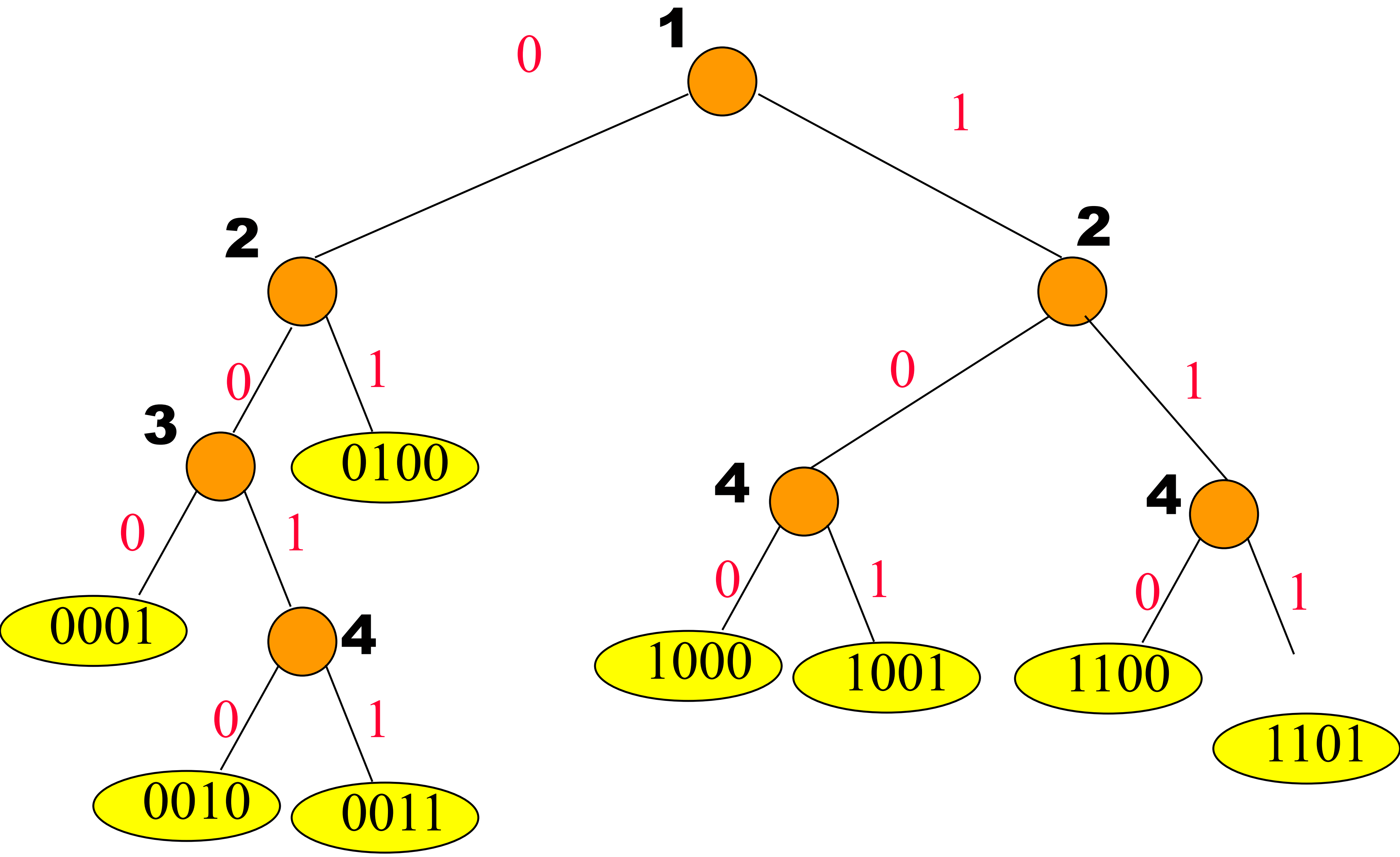
Insert 0010.

Insert



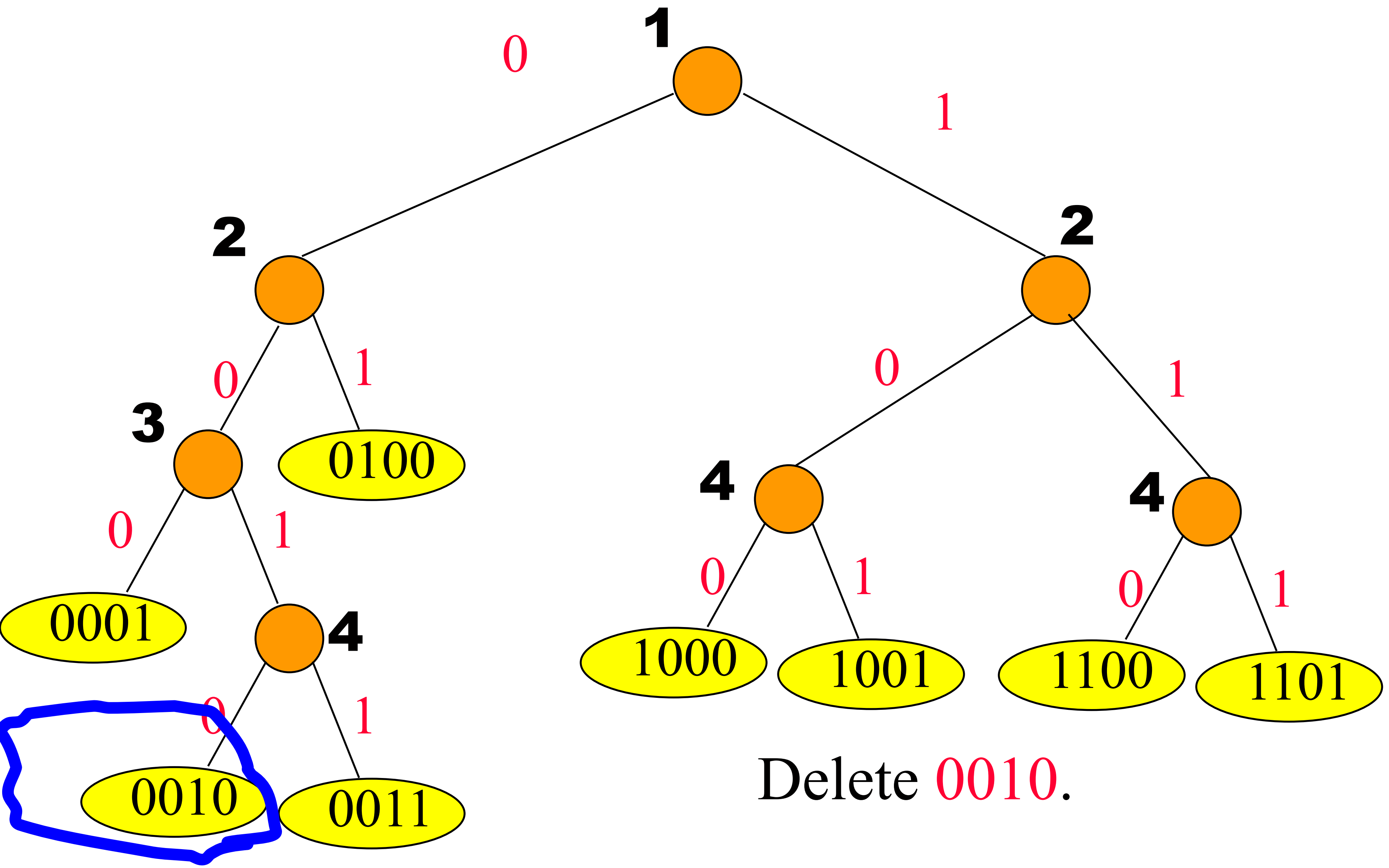
Insert 0100.

Insert



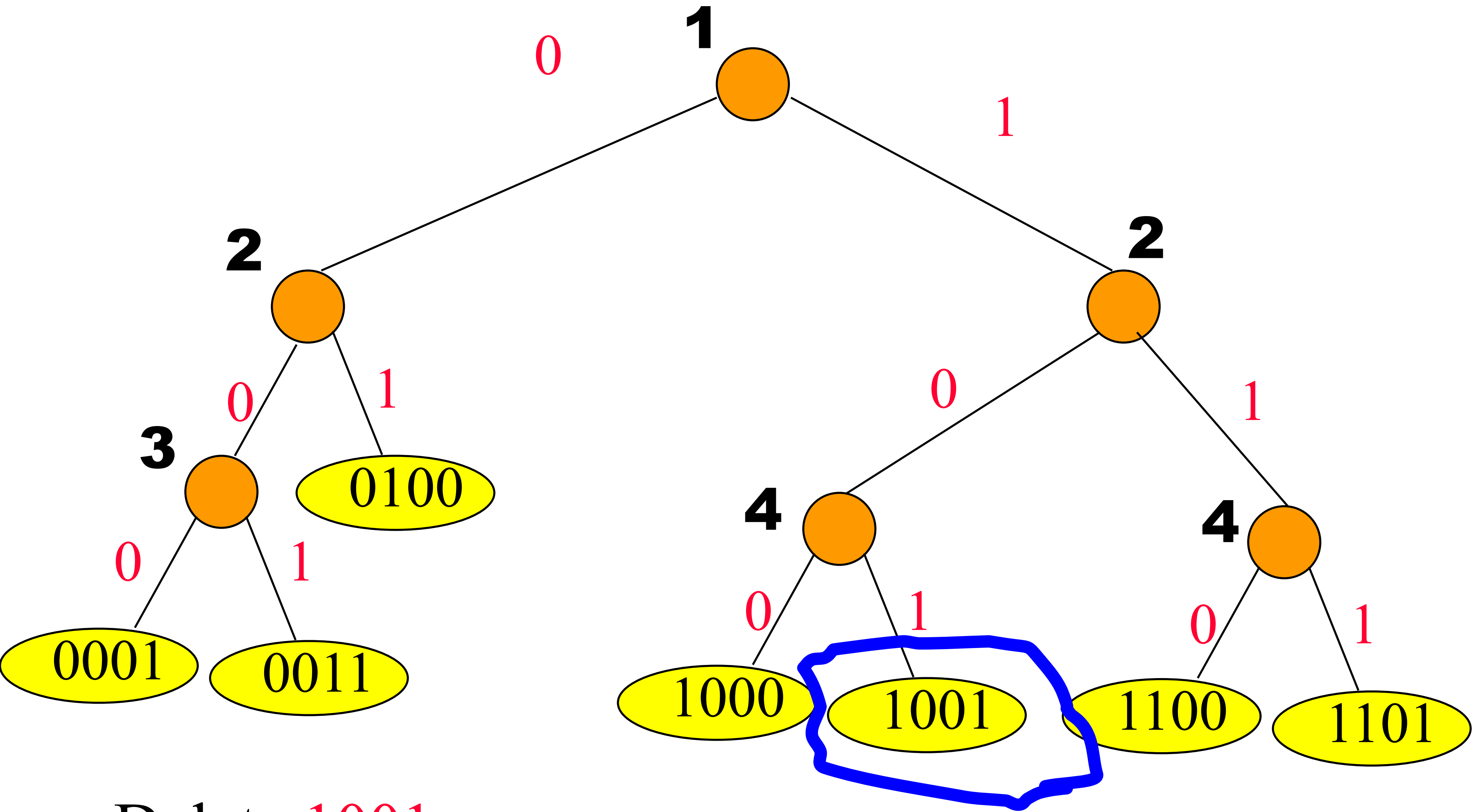
1
0
1
2
2
0
1
0
1
3
0100
4
4
0
1
1
0001
4
0
1
0
1
1000
1001
1100
0
1
1101
0010
0011

Delete



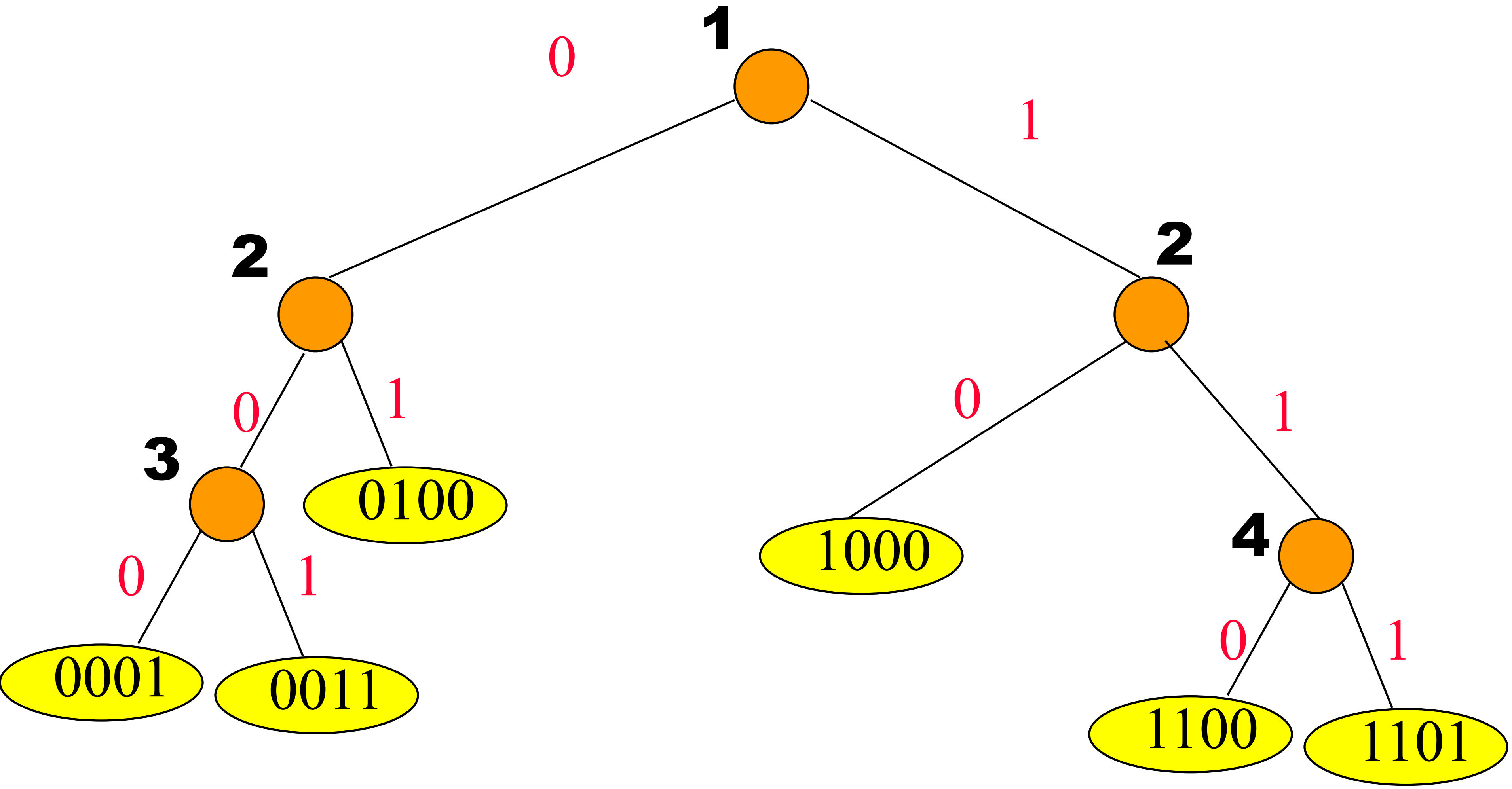
Delete 0010.

Delete



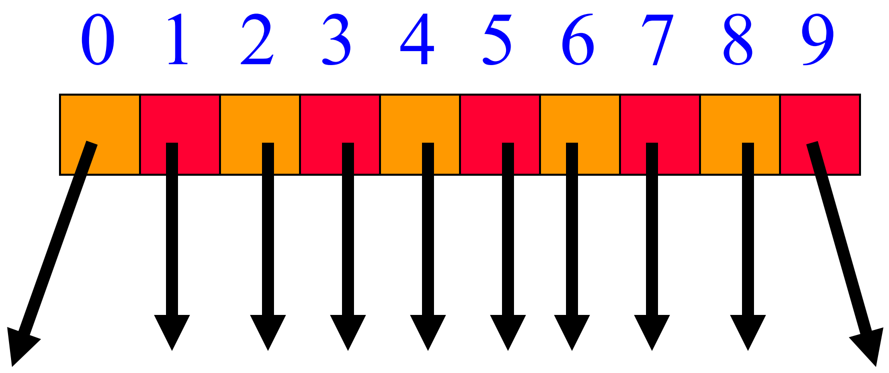
Delete 1001.

Delete



Higher Order Tries

- Key = Social Security Number.
  - 441-12-1135
  - 9 decimal digits.
- 10-way trie (order 10 trie).



Height <= 10.

Social Security Trie

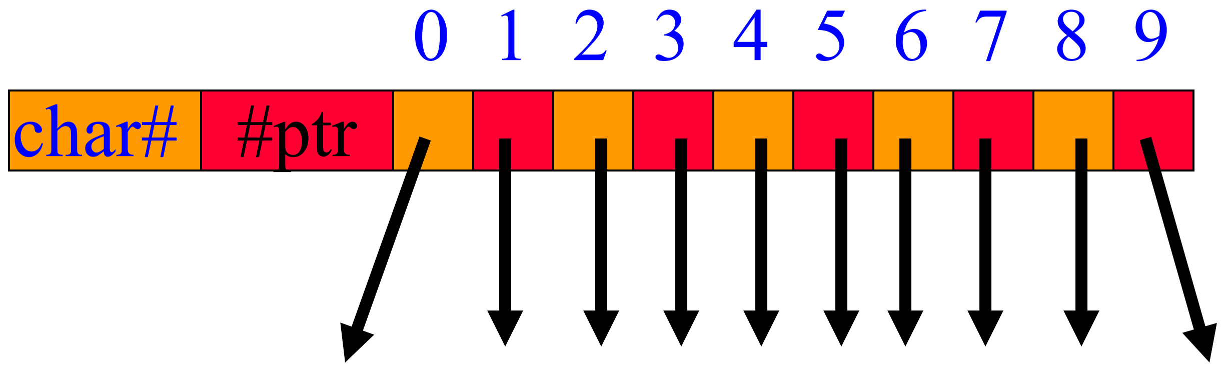
- 10-way trie
  - Height <= 10.
  - Search => <= 9 branches on digits plus 1 compare.
- 100-way trie
  - 441-12-1135
  - Height <= 6.
  - Search => <= 5 branches on digits plus 1 compare.

Social Security AVL & Red-Black

- Red-black tree
  - Height $<= 2\log_2 10^9 \sim 60$.
  - Search => $<= 60$ compares of 9 digit numbers.
- AVL tree
  - Height $<= 1.44\log_2 10^9 \sim 40$.
  - Search => $<= 40$ compares of 9 digit numbers.
- Best binary tree.
  - Height $= \log_2 10^9 \sim 30$.

Compressed Social Security Trie

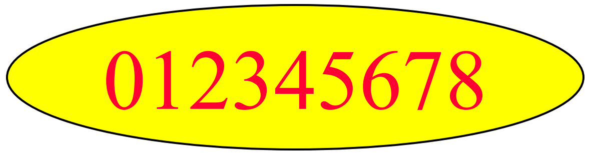
Branch Node Structure



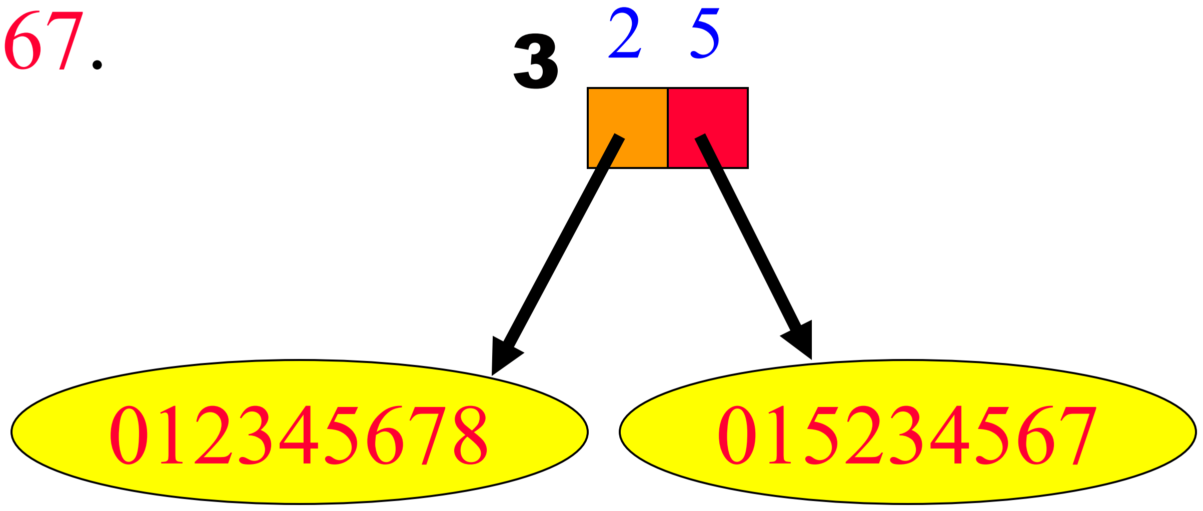
- char# = character/digit used for branching.
  - Equivalent to bit# field of compressed binary trie.
- #ptr = # of nonnull pointers in the node.

Insert

Insert 012345678.

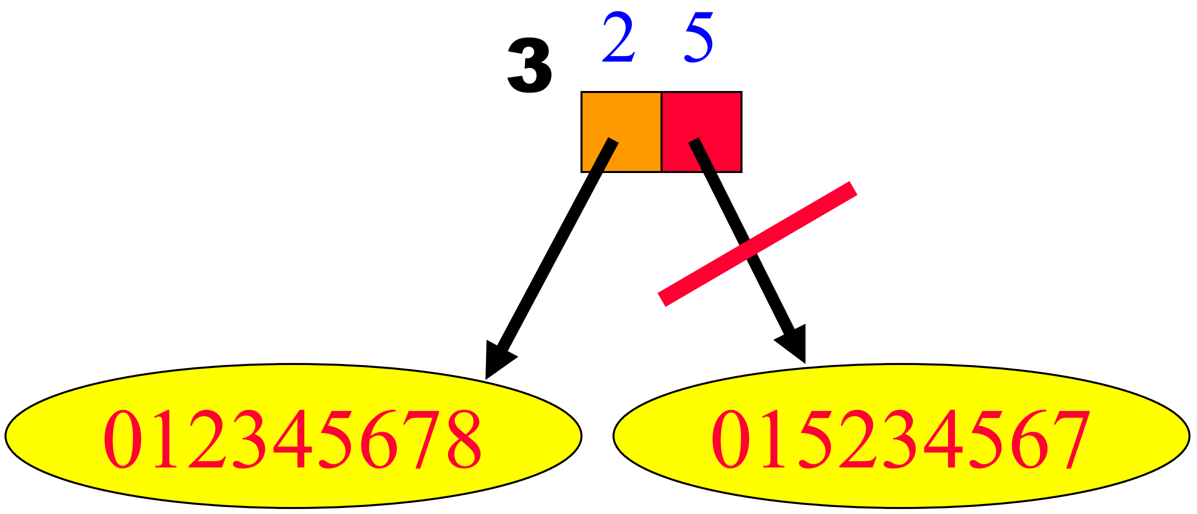


Insert 015234567.



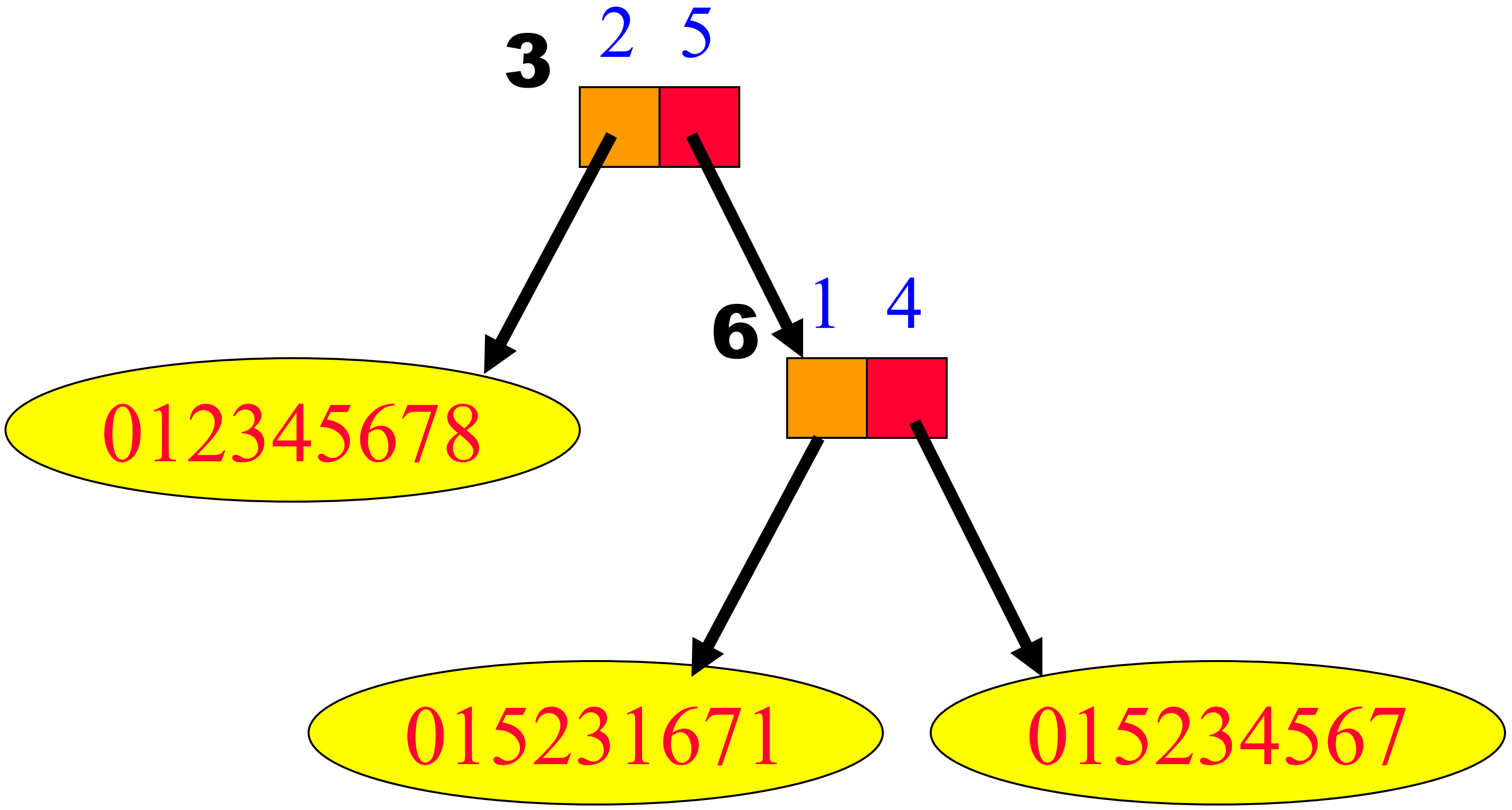
Null pointer fields not shown.

Insert



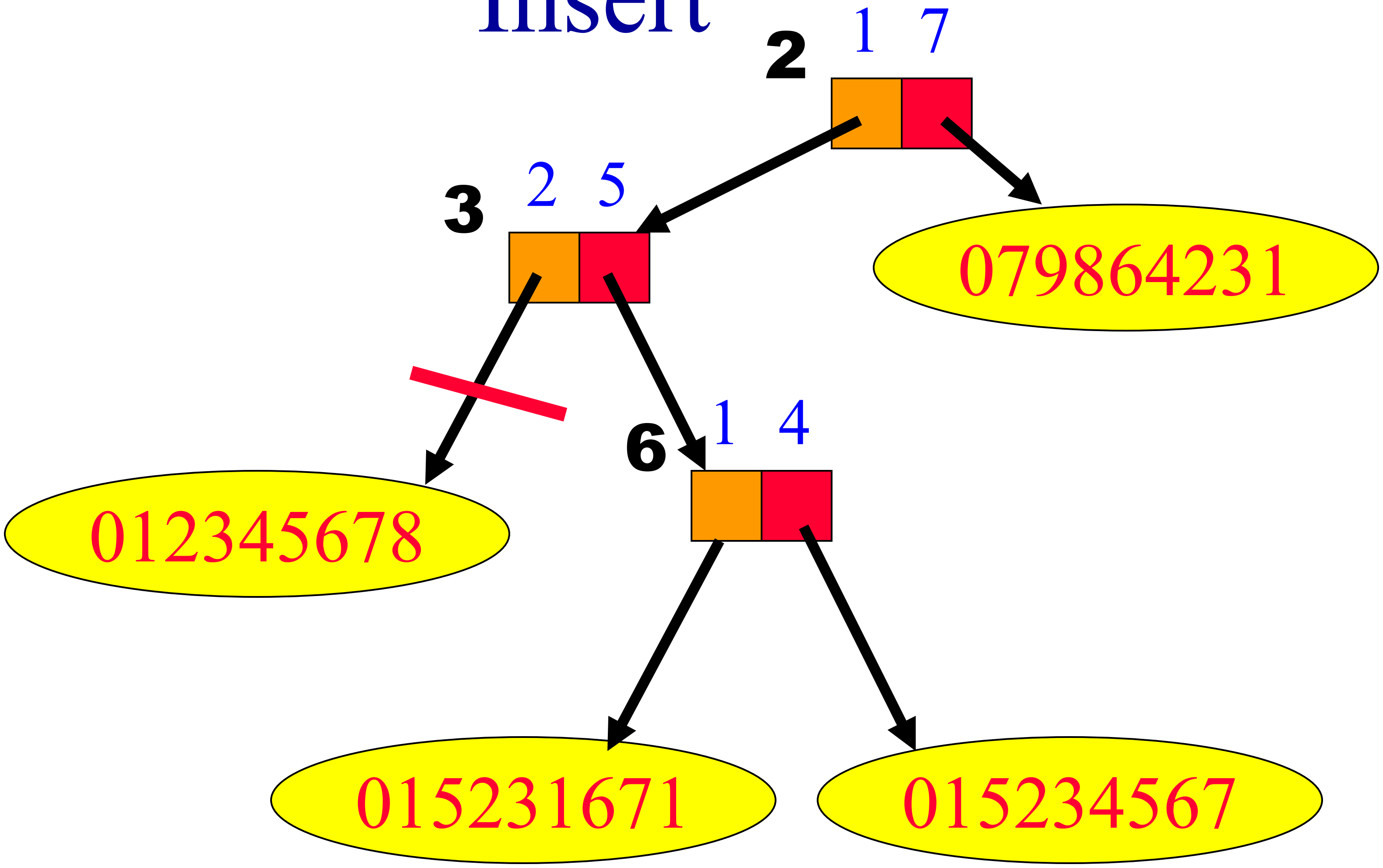
Insert 015231671.

Insert



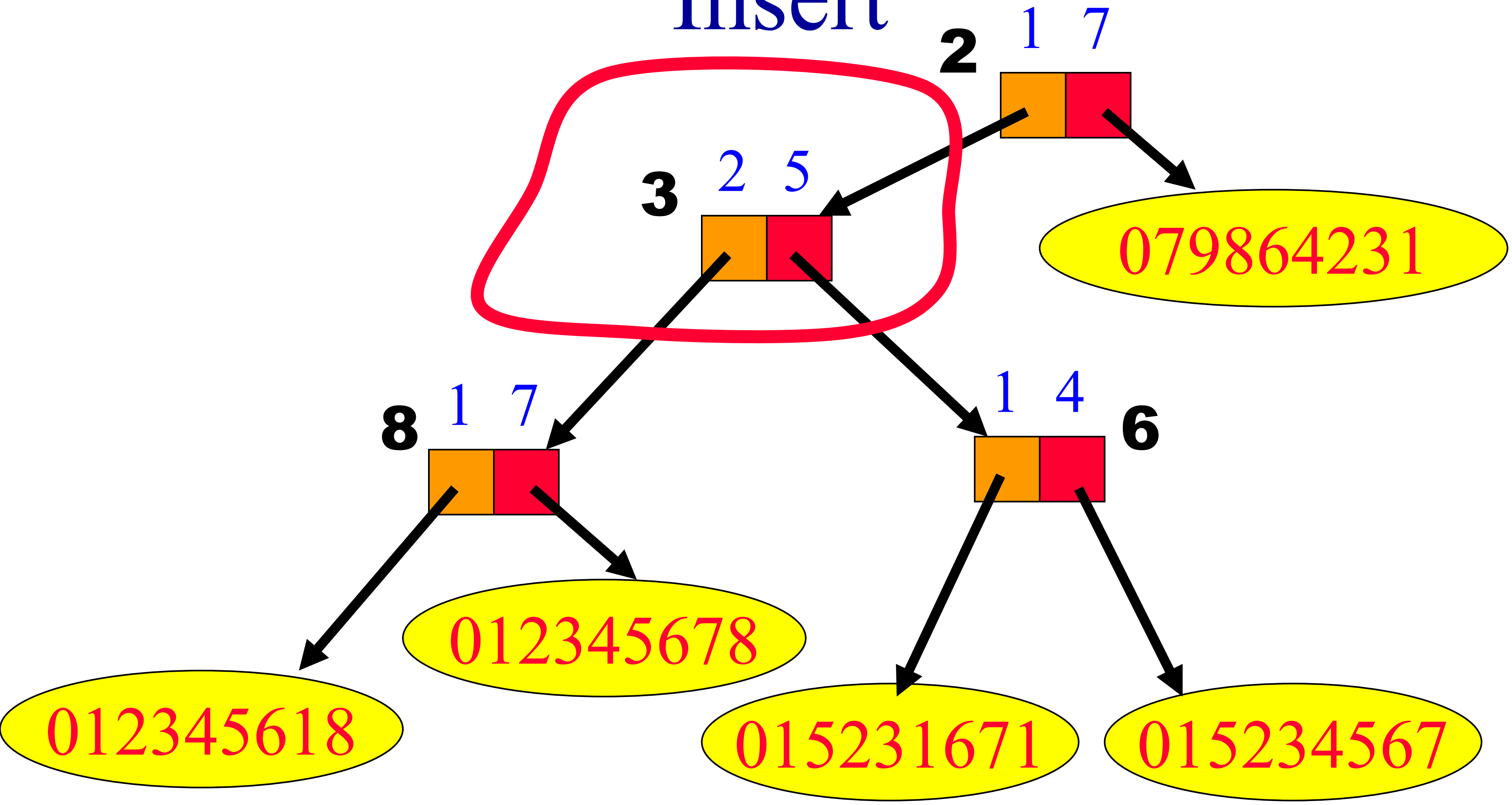
Insert 079864231.

Insert



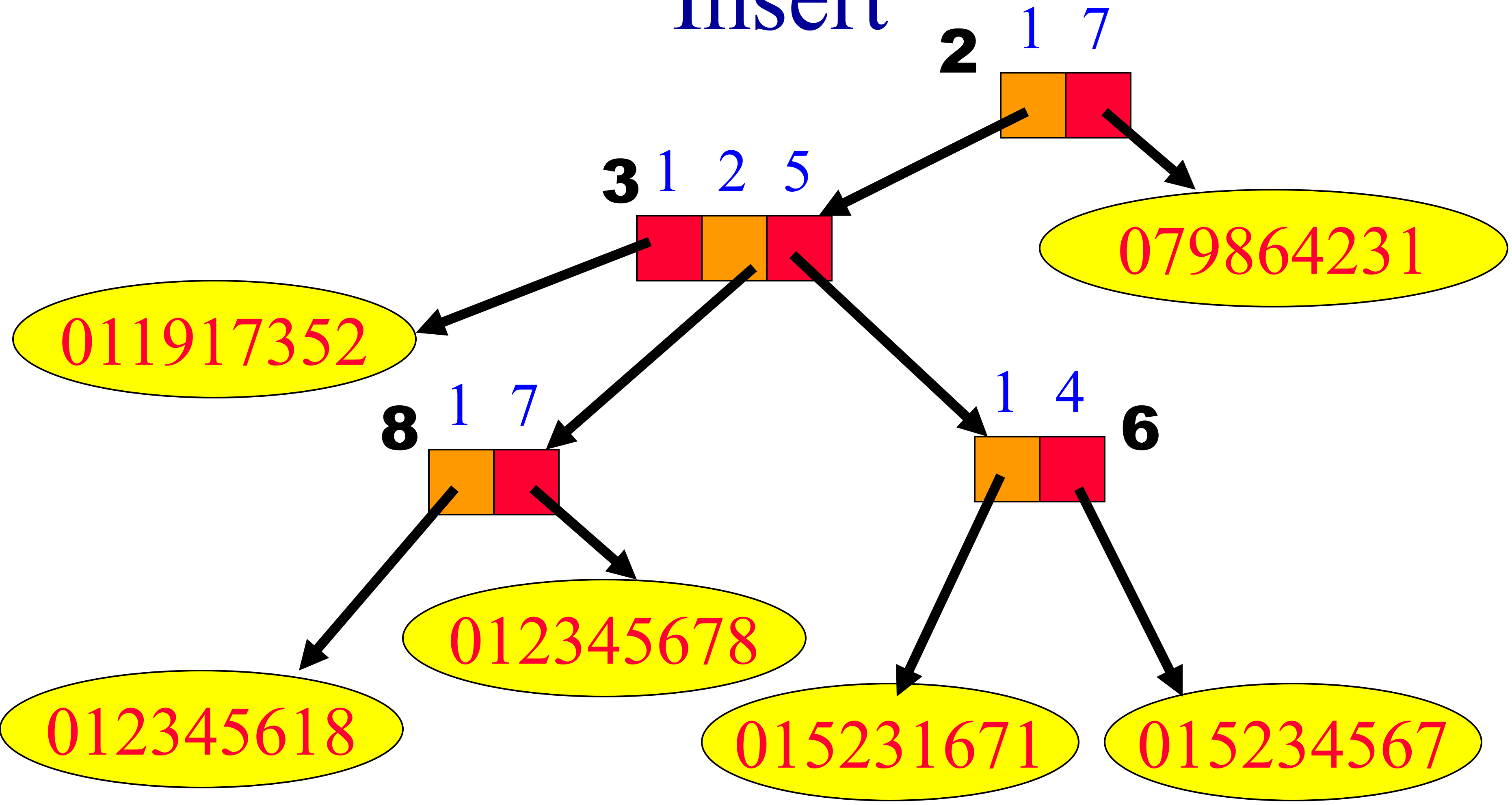
Insert 012345618.

Insert

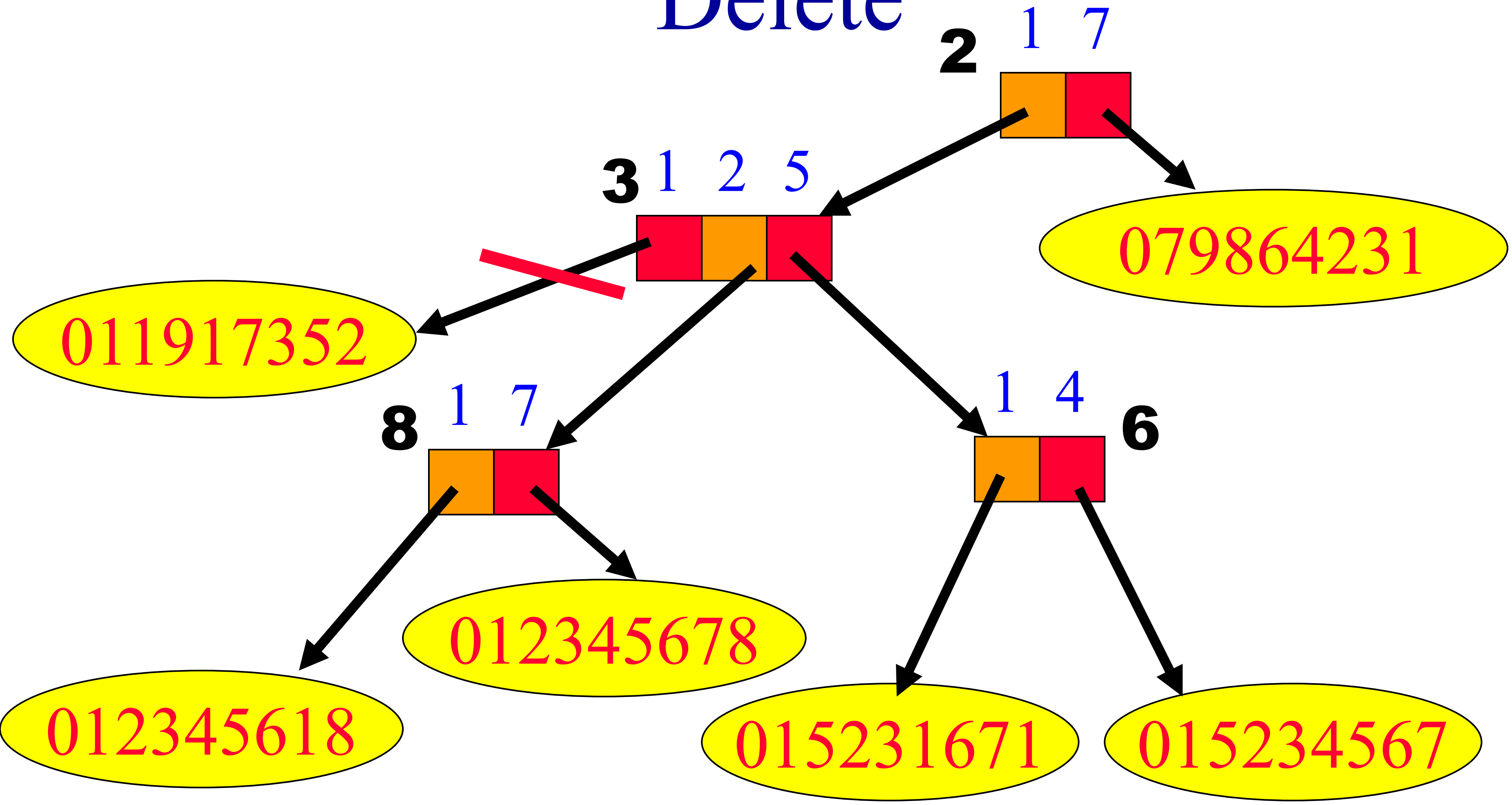


Insert 011917352.

Insert

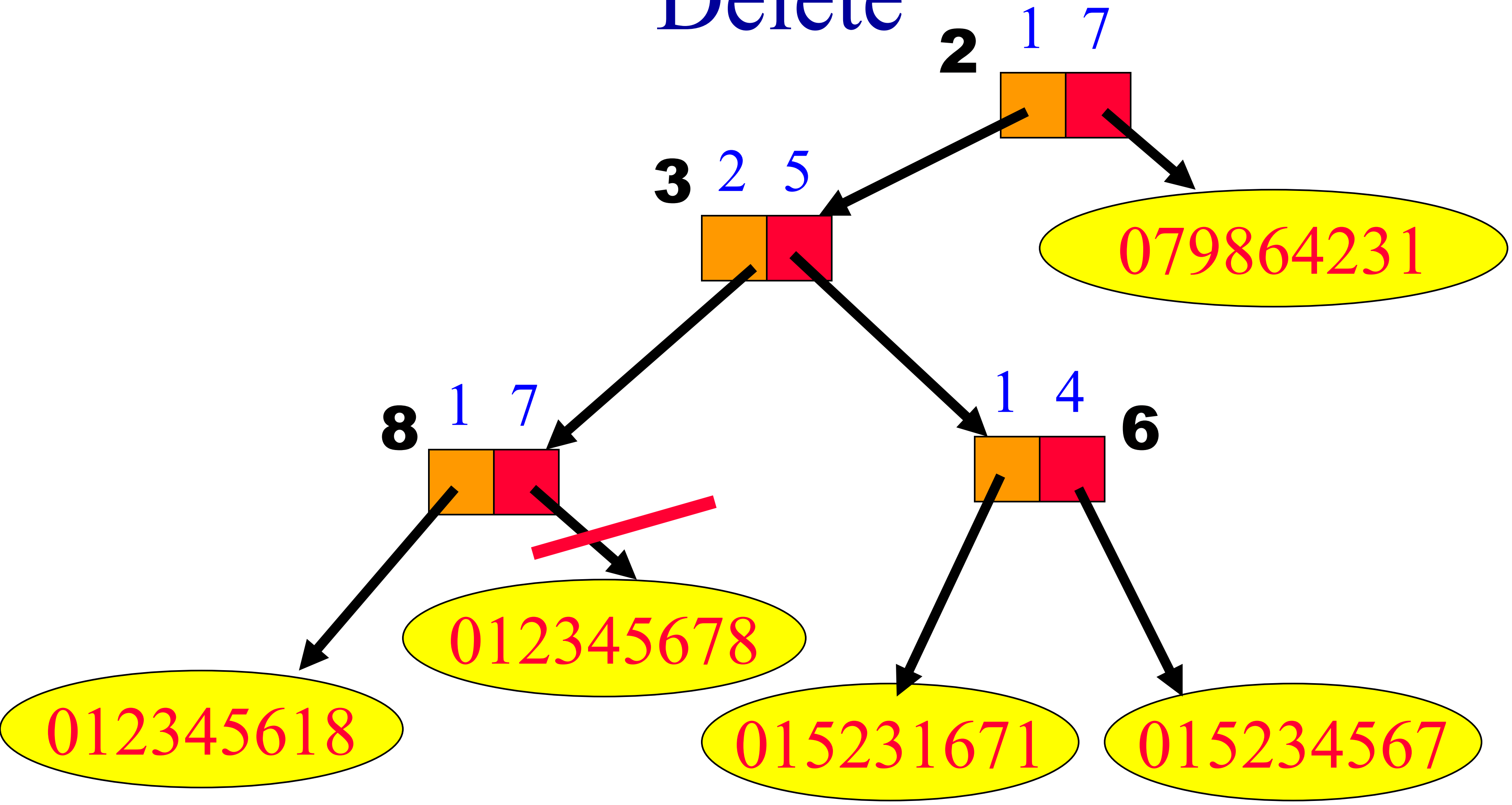

2
1 7
079864231
3
1 2 5
011917352
8
1 7
012345678
012345618
1 4
6
015231671
015234567

Delete



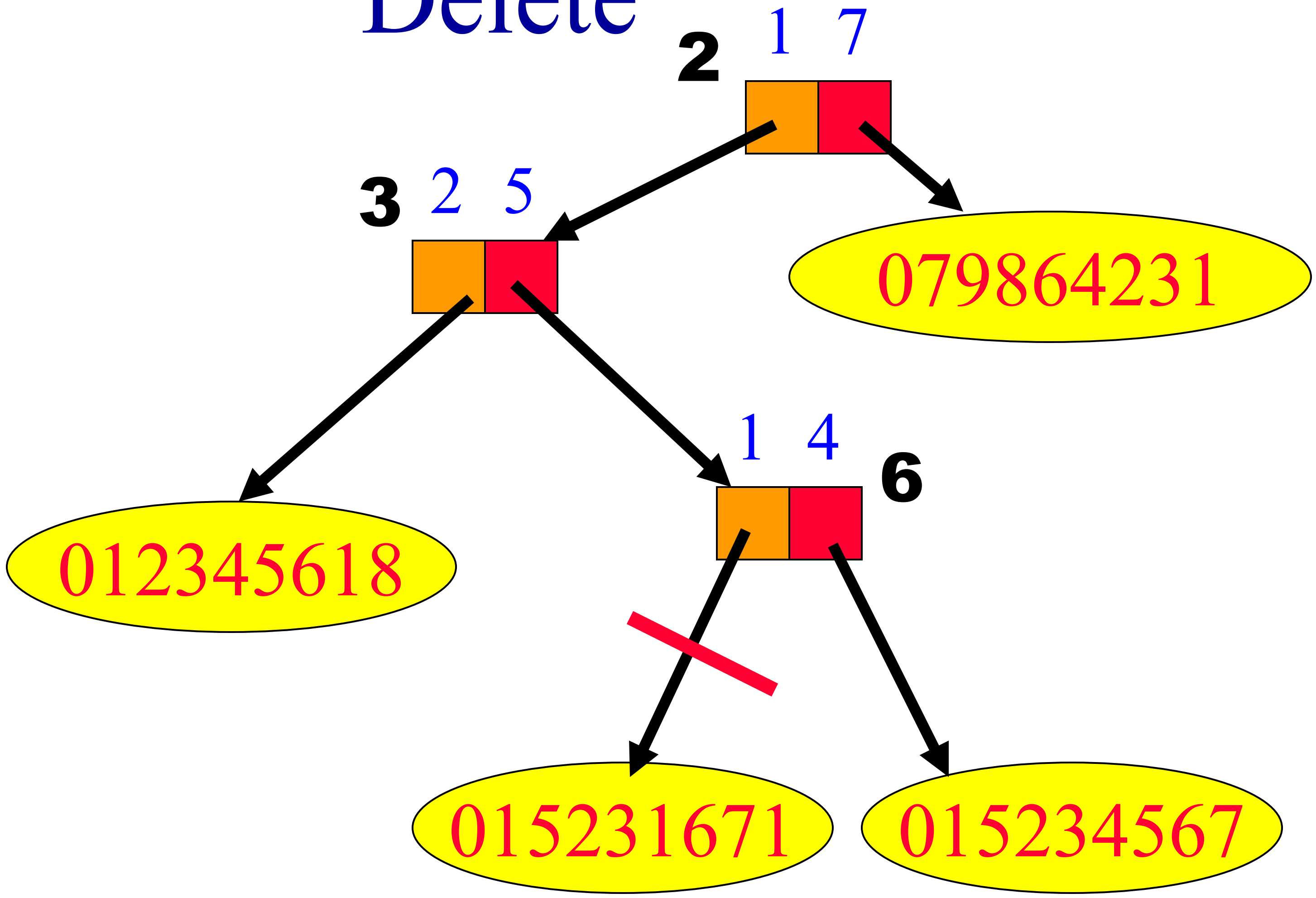
Delete 011917352.

Delete



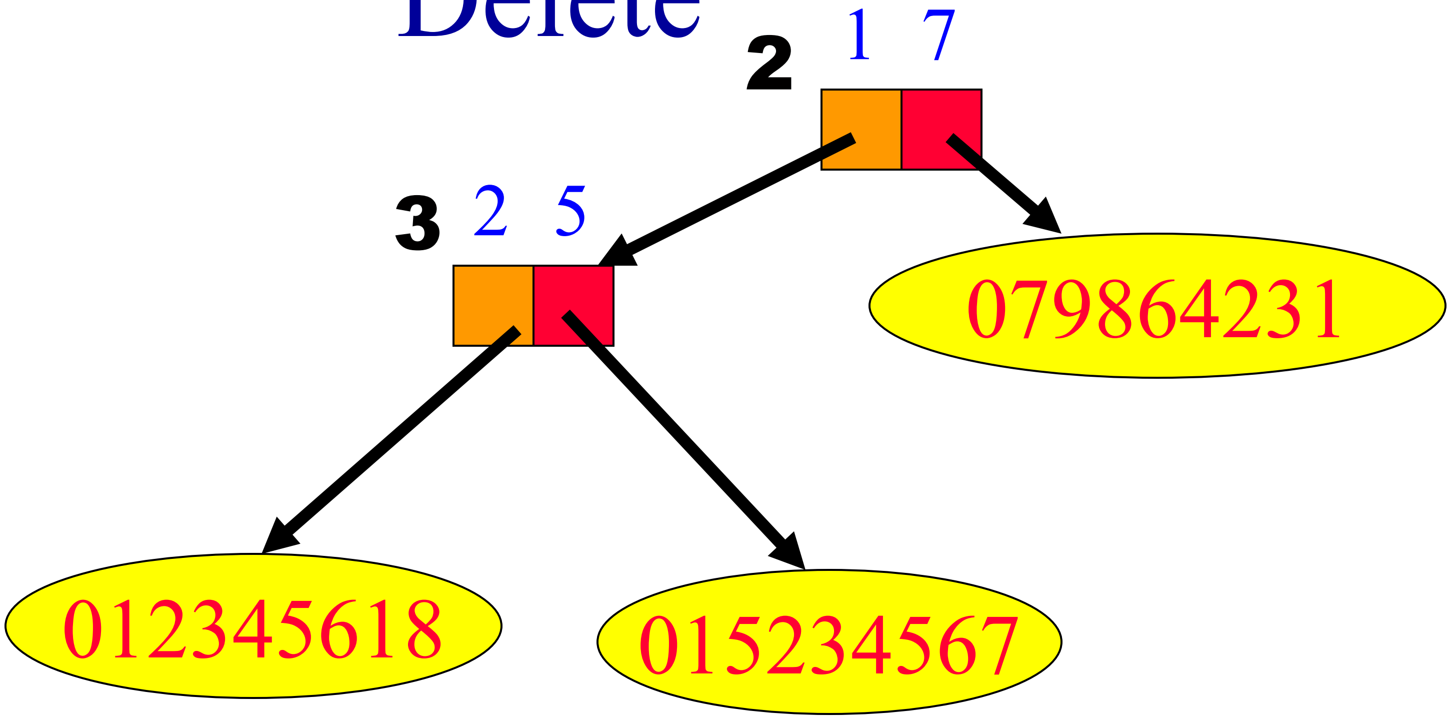
Delete 012345678.

Delete

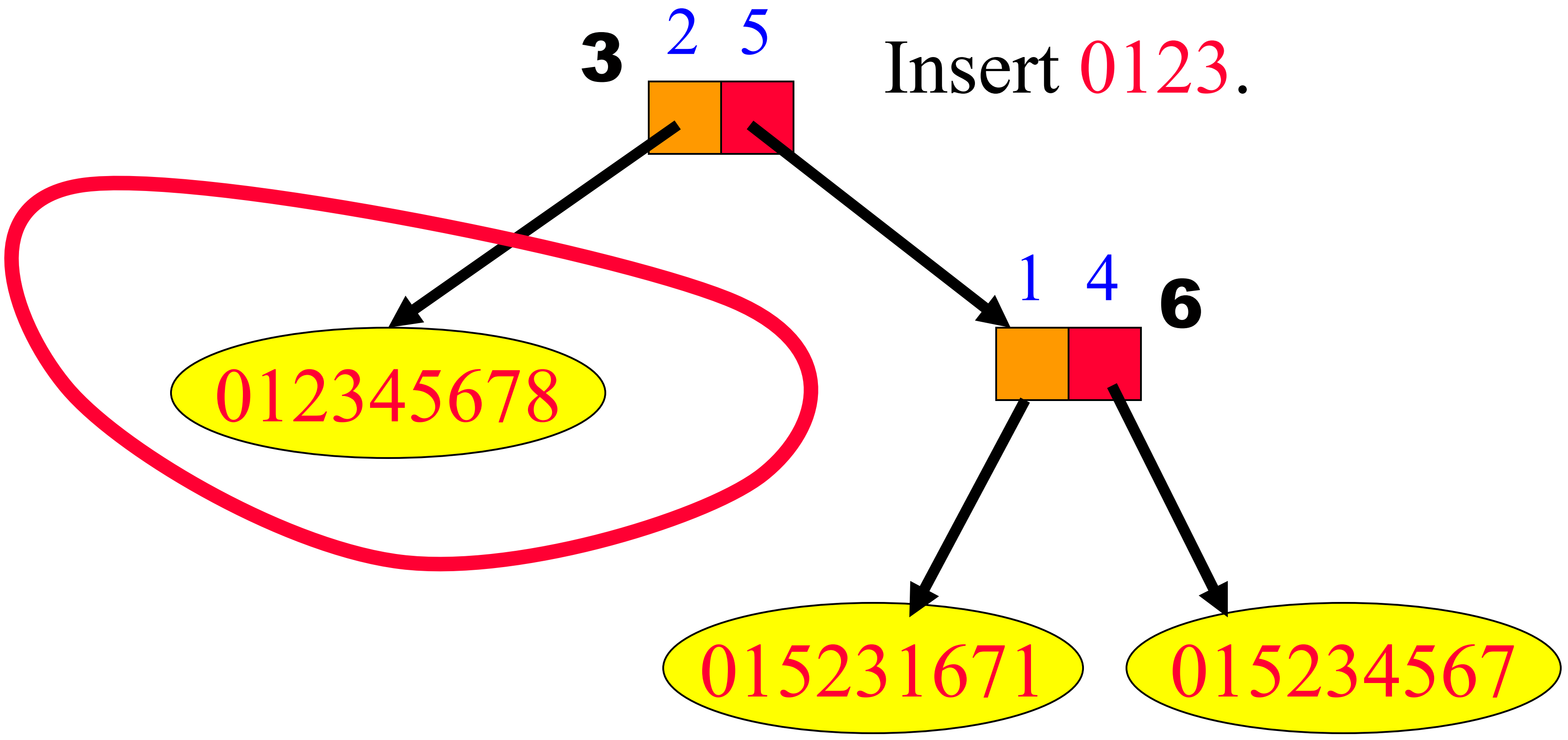


Delete 015231671.

Delete

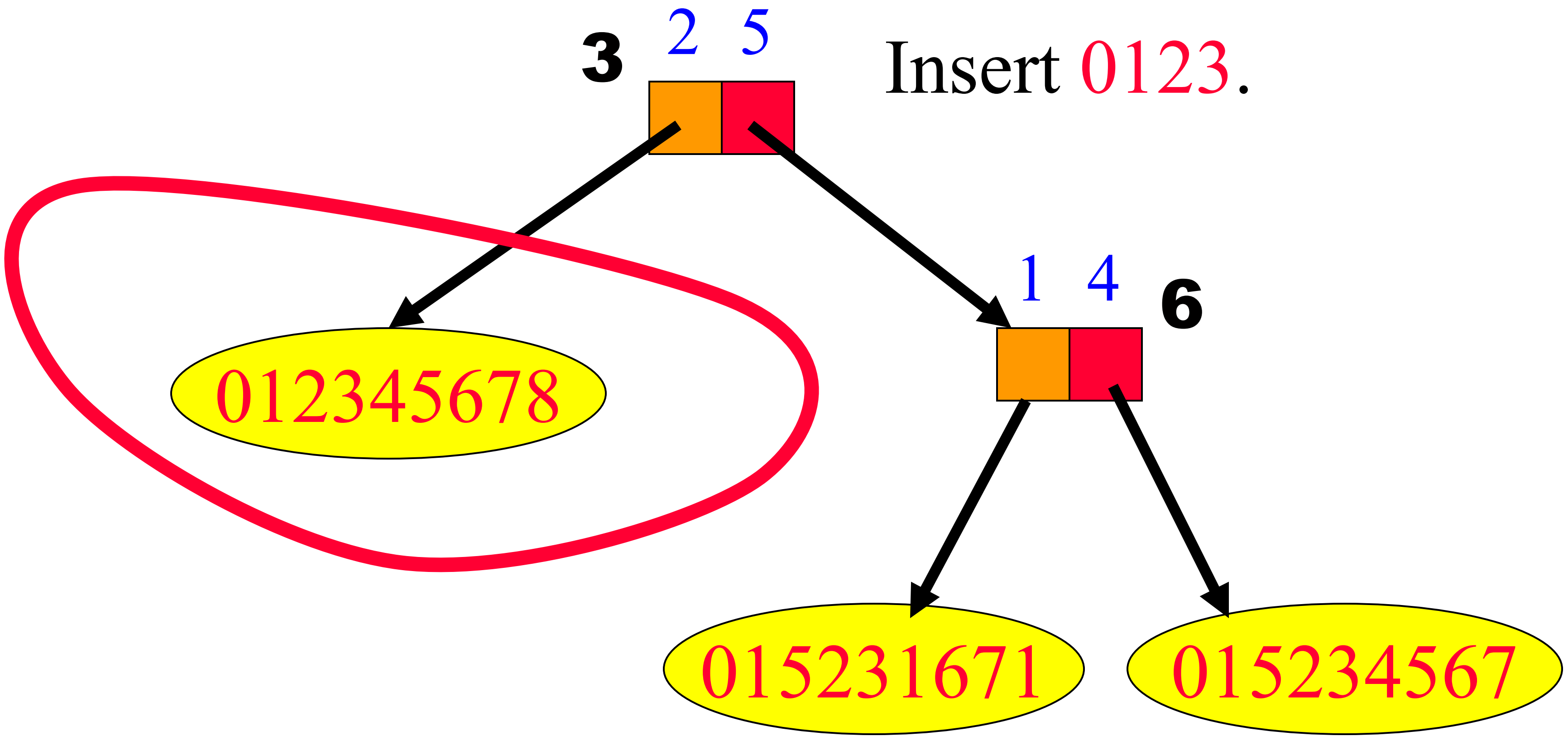


Variable Length Keys



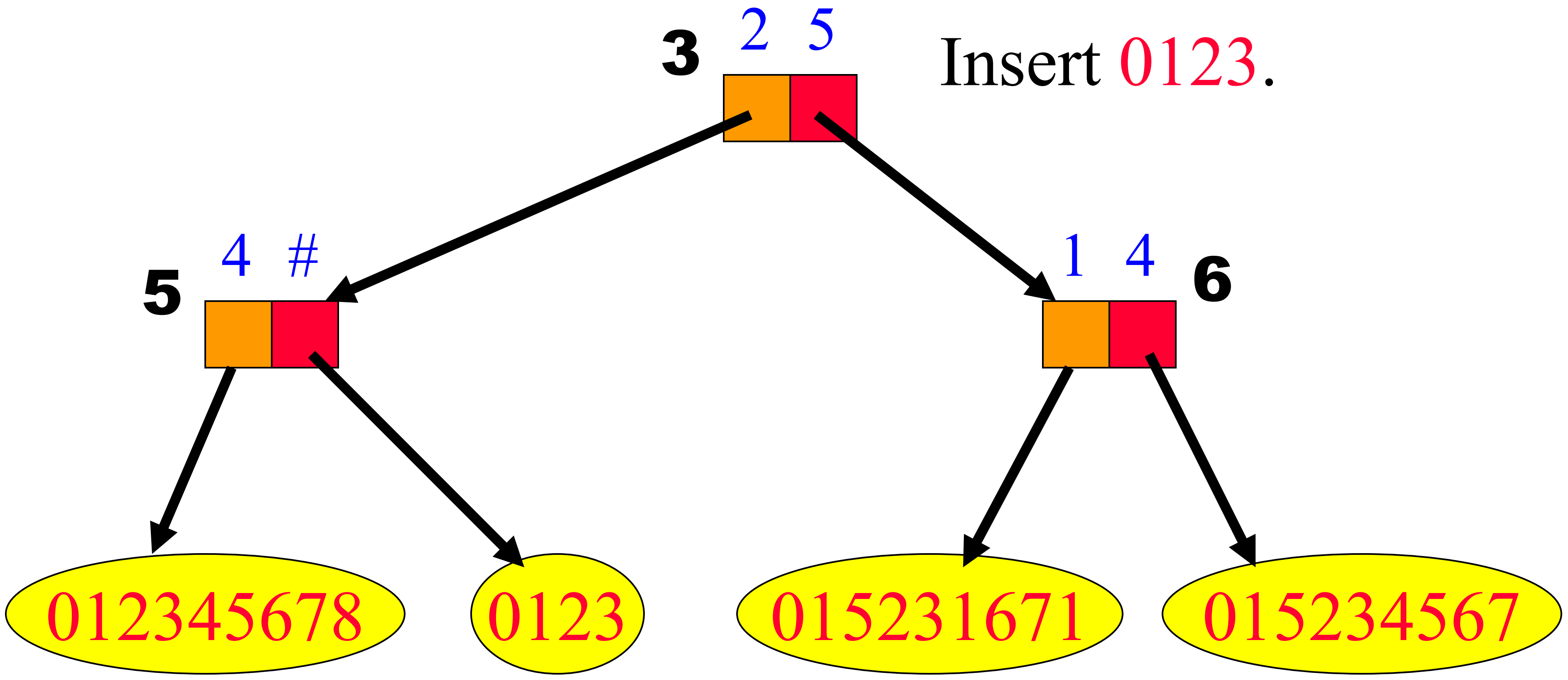
Problem arises only when one key is a (proper) prefix of another.

Variable Length Keys



Add a special end of key character (#) to each key to eliminate this problem.

Variable Length Keys

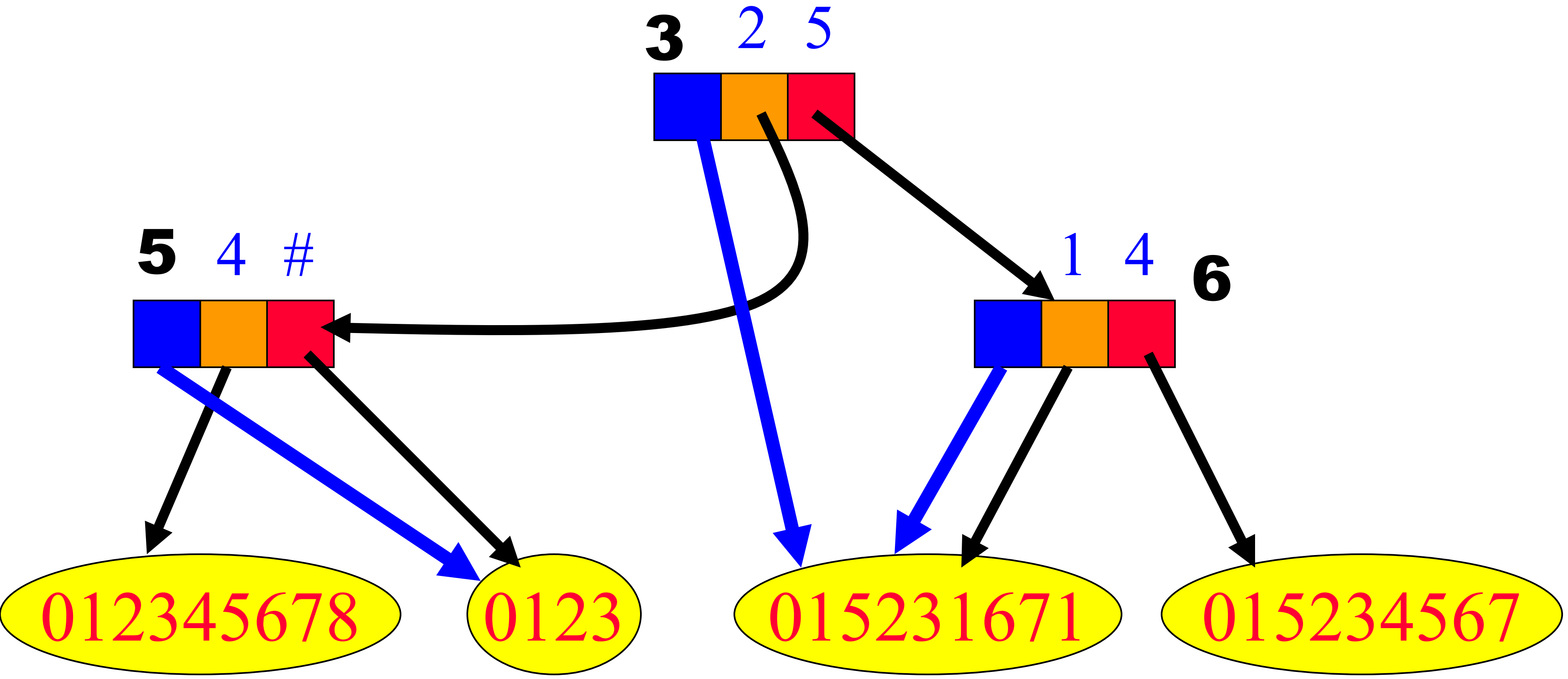


End of key character (#) not shown.

Tries With Edge Information

- Add a new field (element) to each branch node.
- New field points to any one of the element nodes in the subtree.
- Use this pointer on way down to figure out skipped-over characters.

Example



element field shown in blue.