B+ Tree 2

Today's Lecture

- Deletion into B+ tree
- Examples

Deletion Pseudo code

1. Search the key ‘k’ in the tree. Mark the node in the path from root to leaf containing ‘k’. Remove the ‘k’ from leaf.

2. If the leaf ends up with fewer than (M/2 -1) keys, underflow
   a) Adopt data from a neighbor; update the parent
   b) If adopting won’t work, delete node and merge with neighbor, update the parent (delete entry pointing to Leaf)
   c) If the parent ends up with fewer than (M/2 -1) keys, underflow!

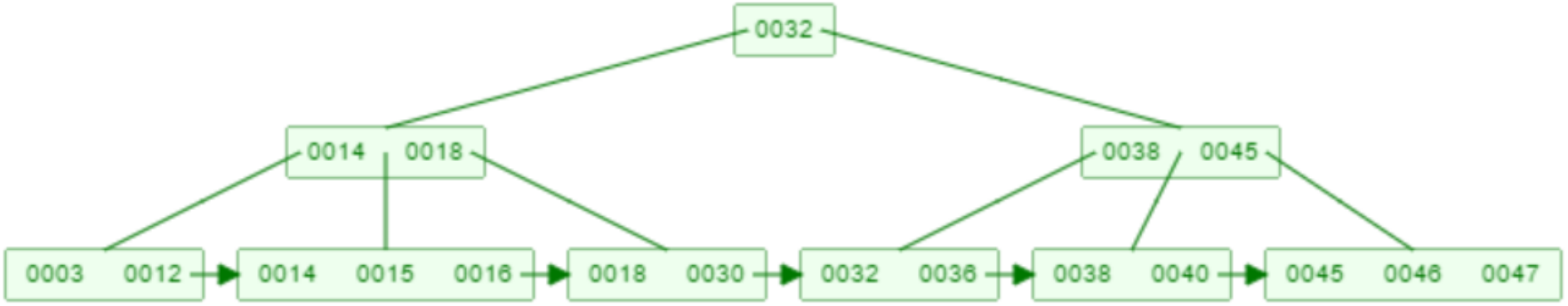
3. If an internal node ends up with fewer (M/2 -1) keys, underflow!
   a) Adopt from a neighbor through parent
   b) If adopting won’t work, delete node and merge with neighbor, update the parent (take the key from parent if its not ‘k’, if it is k then delete it and replace it with appropriate key)
   c) If the parent ends up with fewer than (M/2 -1) keys, underflow!

Deletion Pseudo code

4. If the root ends up with only one child, merge child and root, make resulting node the new root of the tree

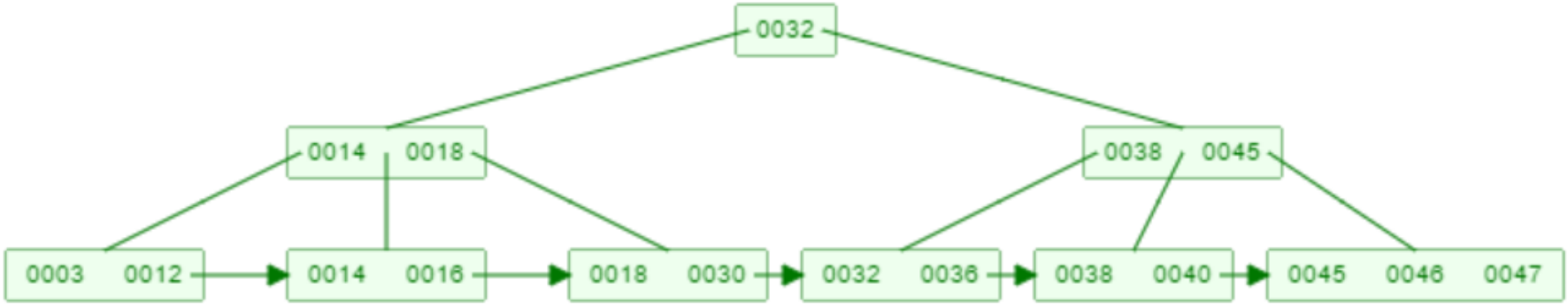
5. If procedure end up without reaching the marked node in first step then replace the ‘k’ in that node with appropriate key

Example


0032
0014
0018
0038
0045
0003
0012
0014
0015
0016
0018
0030
0032
0036
0038
0040
0045
0046
0047

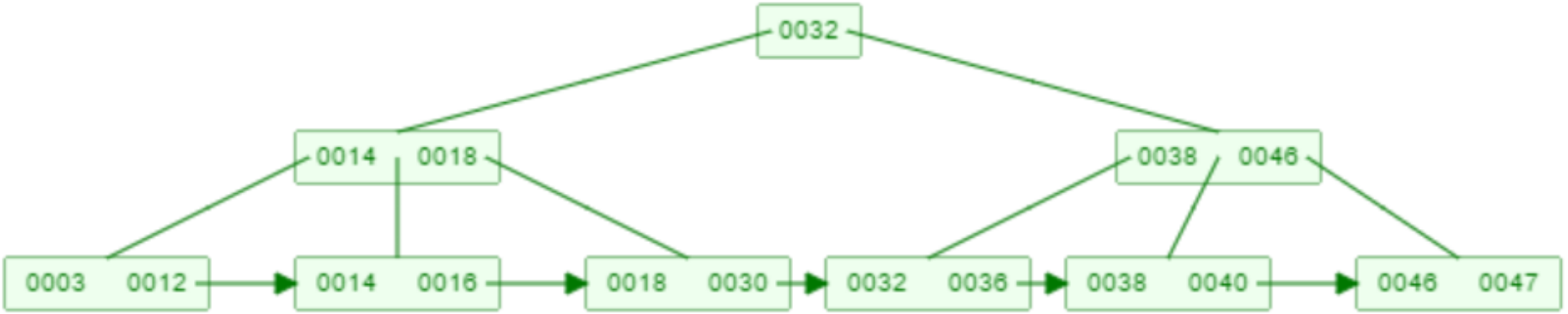
Delete 15

Delete from leaf (parent is not updated)

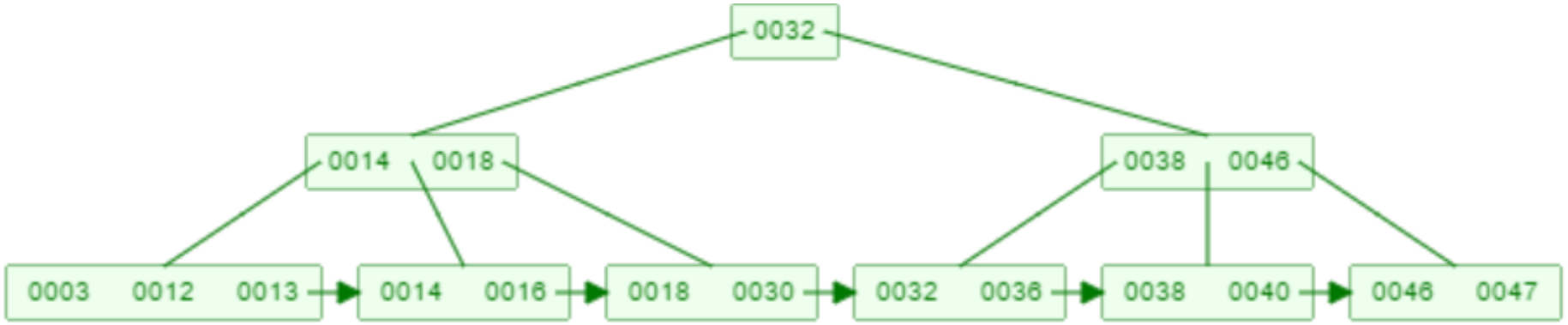


Delete 45

Delete from leaf (parent is updated)

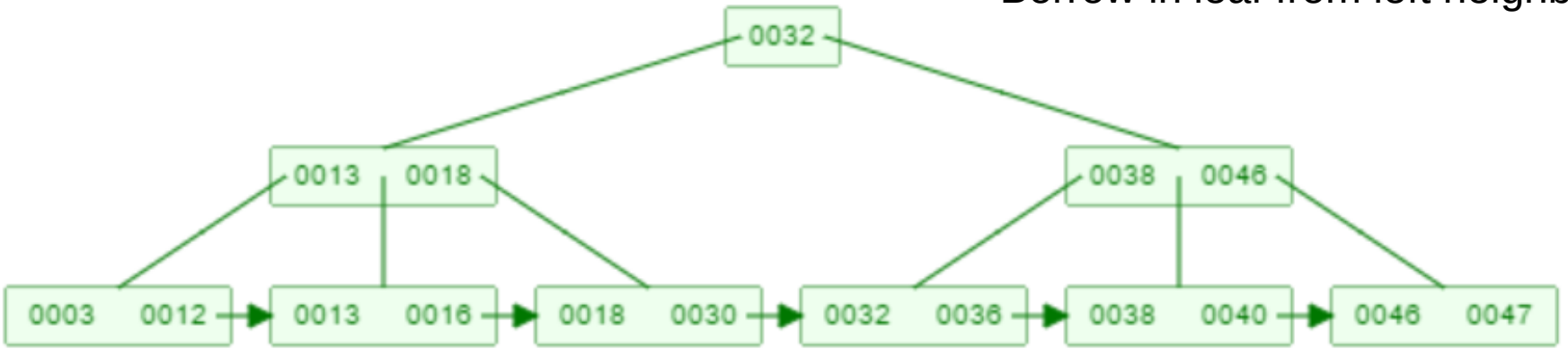


New Tree

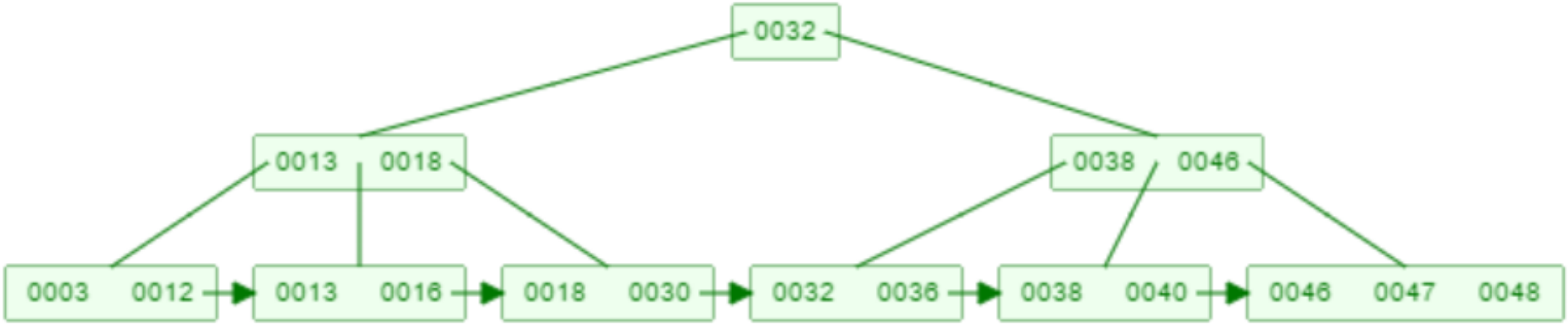

0032
0014
0018
0038
0046
0003
0012
0013
0014
0016
0018
0030
0032
0036
0038
0040
0046
0047

Delete 14

After Deletion
•Leaf Underflow
•Borrow in leaf from left neighbor

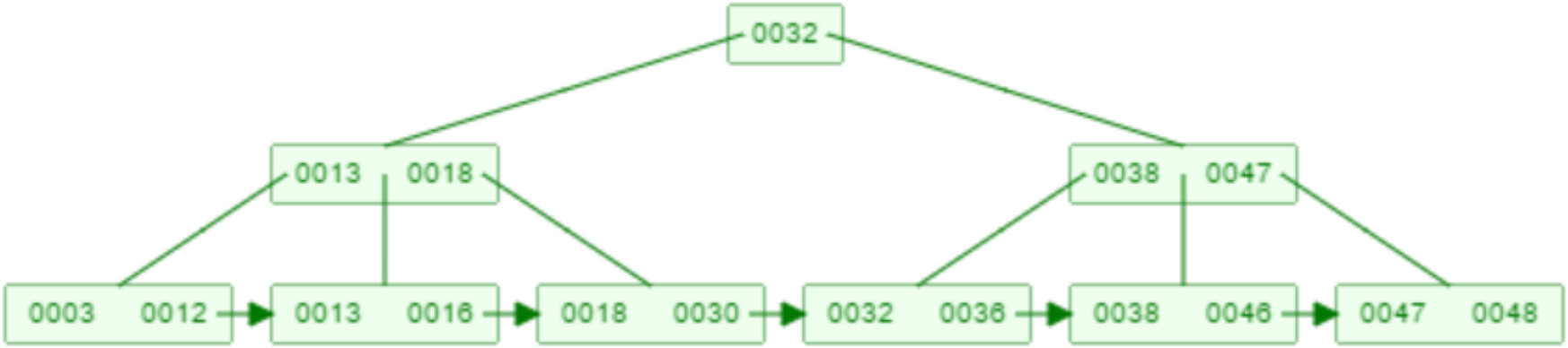


New Tree


0032
0013
0018
0038
0046
0003
0012
0013
0016
0018
0030
0032
0036
0038
0040
0046
0047
0048

Delete 40 ????

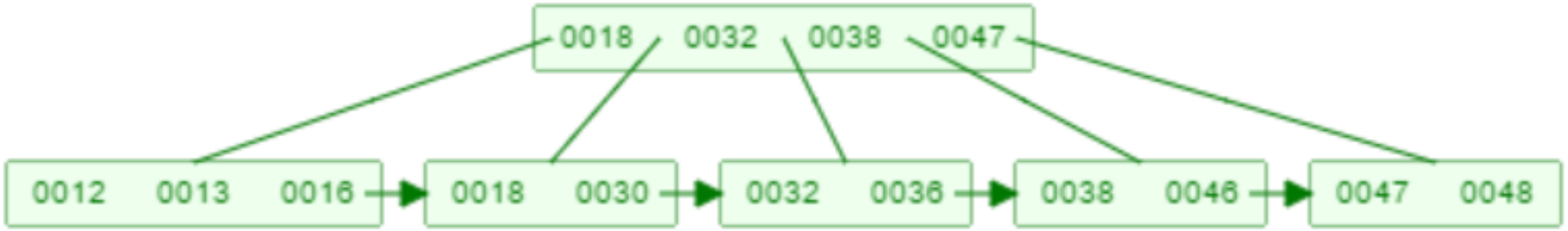
After Deletion
- Leaf Underflow
- Borrow in leaf from right neighbor



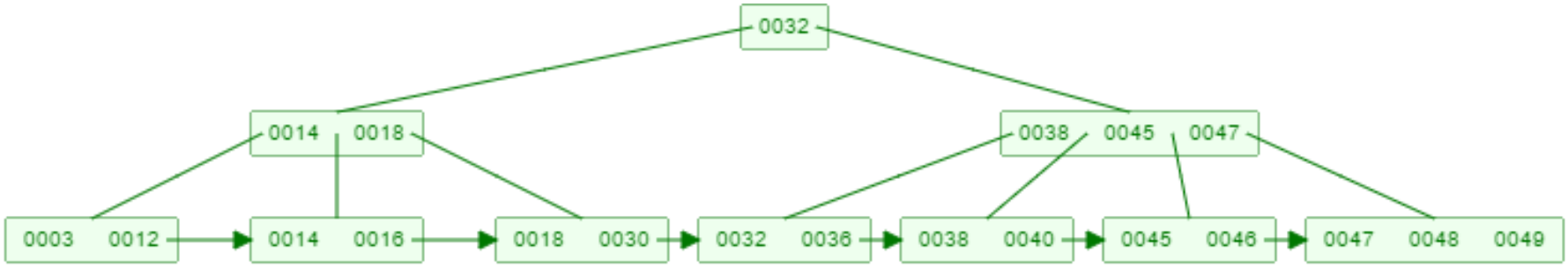
Delete 3

After Deletion

- Leaf underflow
- Parent (Internal node) underflow
  -> merge



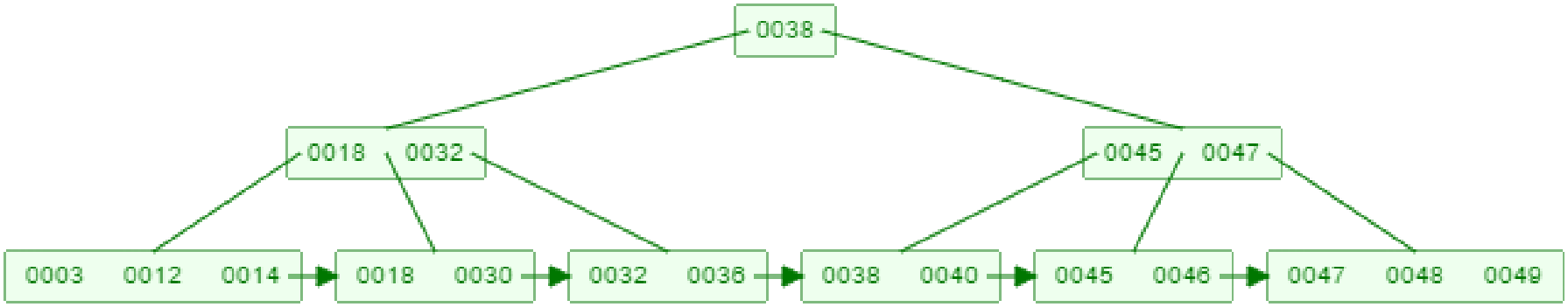
New Tree


0032
0014 0018
0038 0045 0047
0003 0012
0014 0016
0018 0030
0032 0036
0038 0040
0045 0046
0047 0048 0049

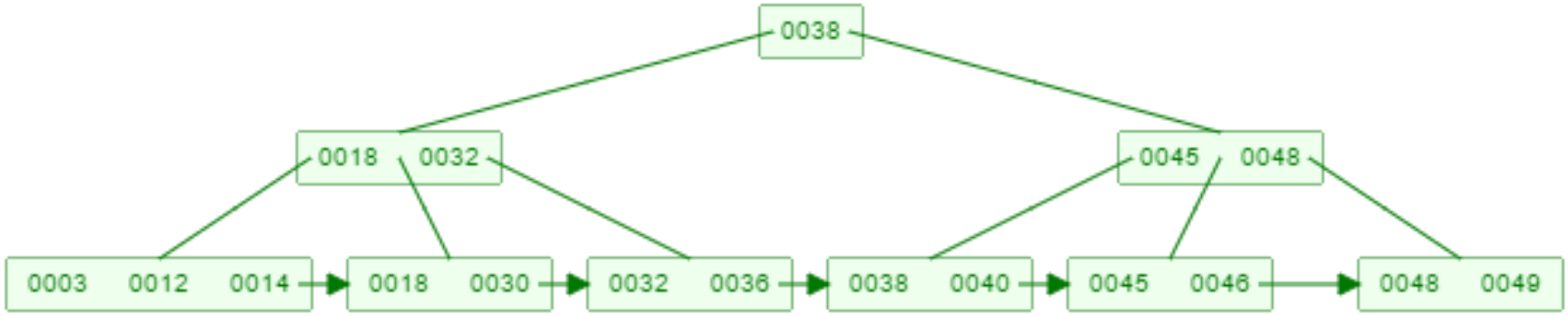
Delete 16

After Deletion

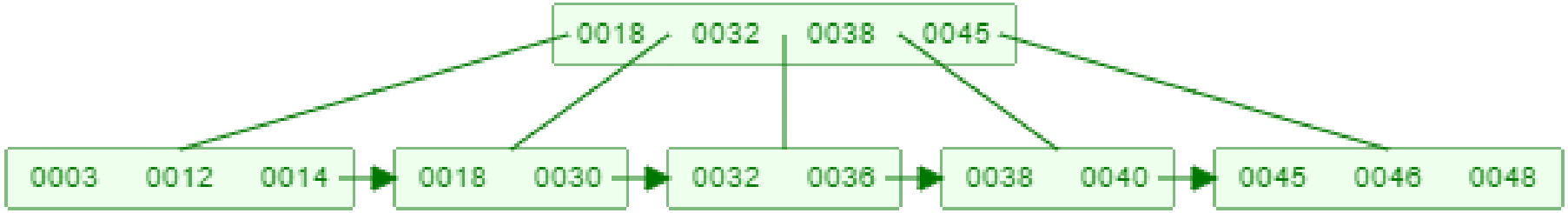
- Leaf underflow
- internal node underflow
  -> borrow



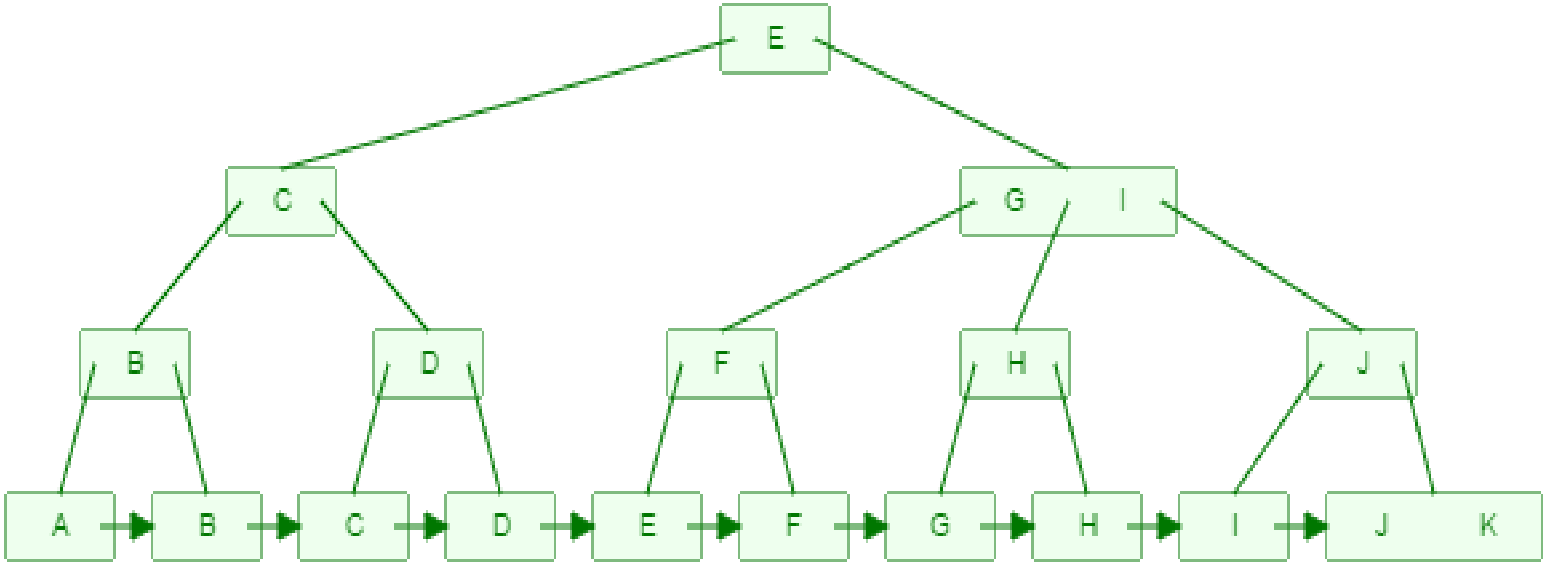
Delete 47


0038
0018
0032
0045
0048
0003
0012
0014
0018
0030
0032
0036
0038
0040
0045
0046
0048
0049

Delete 49

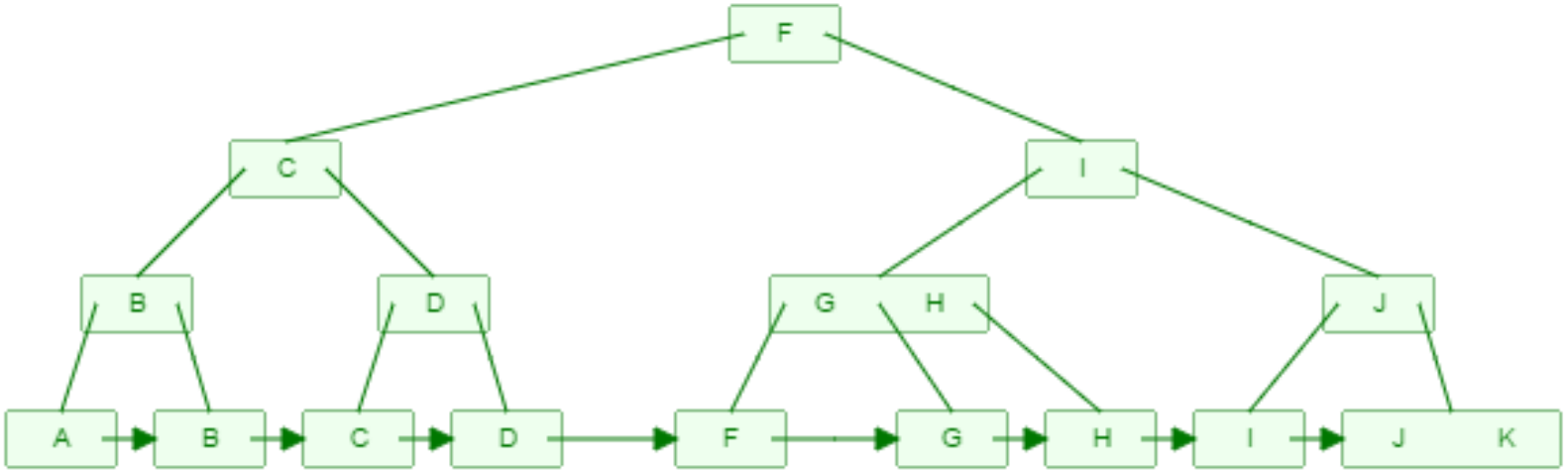

0018 0032 0038 0045
0003 0012 0014
0018 0030
0032 0036
0038 0040
0045 0046 0048

Task



- Delete E

Task



- Delete I
- Delete C

Analysis

- $O(t \log_t n)$

Task

- Write a pseudocode to search data in B+tree?

```
BplusSearch (x,k)
{i=1;
While (i≤x.n && k≥ x.key_i)
   if(x.leaf !=True  & x!=x.key_i)  {i=i+1}
If(i≤x.n && x.leaf &k==x.key_i)
    return (x,i)
else if x.leaf
Return nill
else
  BplusSearch(x.c_i, k)
}
```

Summary

- Analysis : similar to B tree accept disk accesses.

- The **leaf nodes of B+ trees are linked**, so doing a full scan of all objects in a tree requires just one linear pass through all the leaf nodes. A B tree, on the other hand, would require a traversal of every level in the tree.

- B+ trees don't have data associated with interior nodes, **more keys can fit on a main memory**.

- ***Range or sequential search:*** Providing support for range queries in an efficient manner.

- Because B trees contain data with each key, nodes closer to the root, can be accessed more quickly.

Summary

- B+ tree is used for an obvious reason and that is search speed. As we know that there are space limitations when it comes to memory, and not all of the data can reside in memory, and hence majority of the data has to be saved on disk. Disk as we know is a lot slower as compared to memory because it has moving parts.