B-Trees

Deletion and Merge

Drawn by Rohan Kadkol using Dr. Duncan’s handout and guidance

B-Trees Operations

- Delete
  - Merge

Some important points on deletion

- Deletion **always** starts in the **leaf**
- When deleting an entry not in a leaf node, find it’s replacement in a leaf node using the indicated strategy

Some important points on deletion

- Two strategies:
  - In-order successor
  - In-order predecessor
- If a strategy isn’t specified, assume In-order successor
- Priority order:
  - Specified strategy (First)
  - The other strategy (If the first is not possible)

Some important points on deletion

- If the hole is in a leaf
  - Eliminate it if the leaf has an extra key
  - Fill it if it doesn’t
- If the hole is eliminated at the root, the tree becomes shorter by 1.
- If the hole is eliminated at a non-leaf level, and is not a root, it creates an orphan that has to be adopted.

Some important points on deletion

- Two cases to fill hole:
  - Easy: Sibling has an excess key.
  - Complex: Sibling doesn’t have an extra key.
    - You’ll need to merge the siblings and transfer the hole to the parent.
    - After the hole is filled, orphans will need to be accomodated.

Some important points on deletion

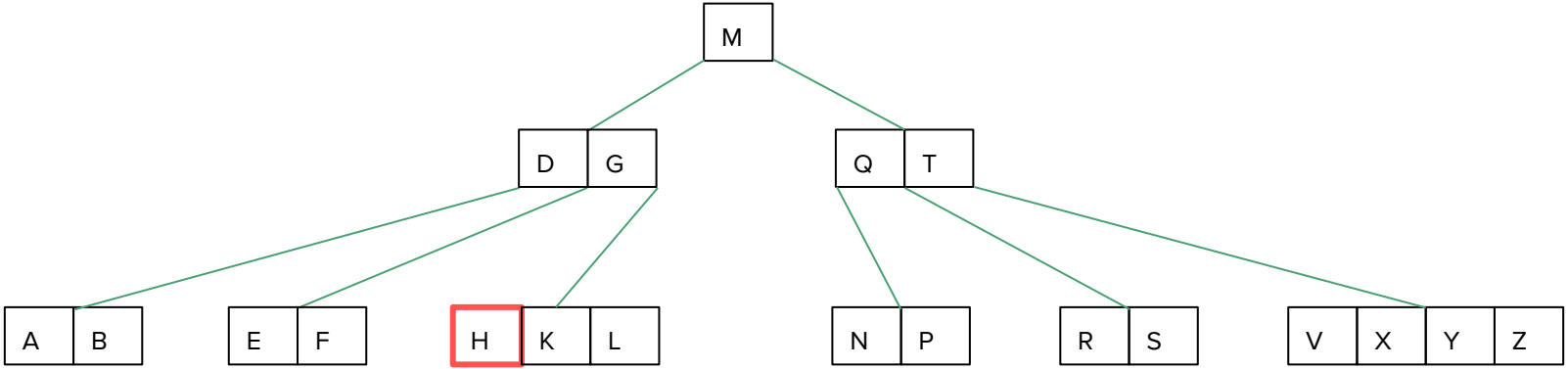
- We only merge when both siblings have the minimum number of entries. ie, $\lceil m/2 \rceil - 1 + \lceil m/2 \rceil - 1 = (m-1)$. Hence, we won't exceed the maximum
  - Eg. 5th order has at most 4 entries
  - We merge when the siblings have 2 entries each and the parent has a hole.

Some important points on deletion

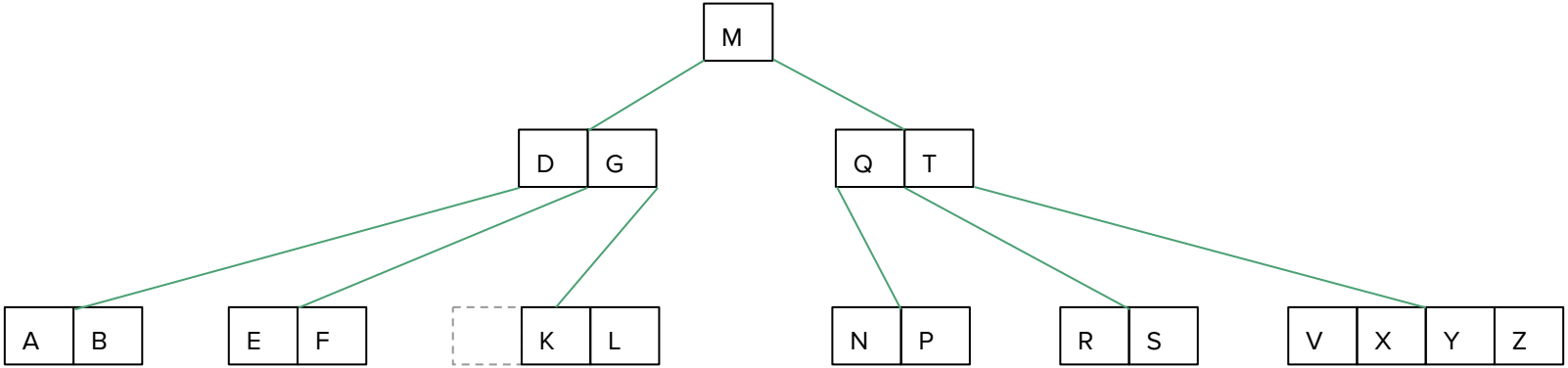
- Whenever we merge, we always look at the same level (in the same node) or upwards.
- During adoption: Left child of right subtree becomes the right child of the left subtree, and vice versa.
- When hole,
  - Eliminate, if possible (First priority)
  - Else, fill it using the specified strategy. (Second priority)

B-Tree Deletion
Example 1

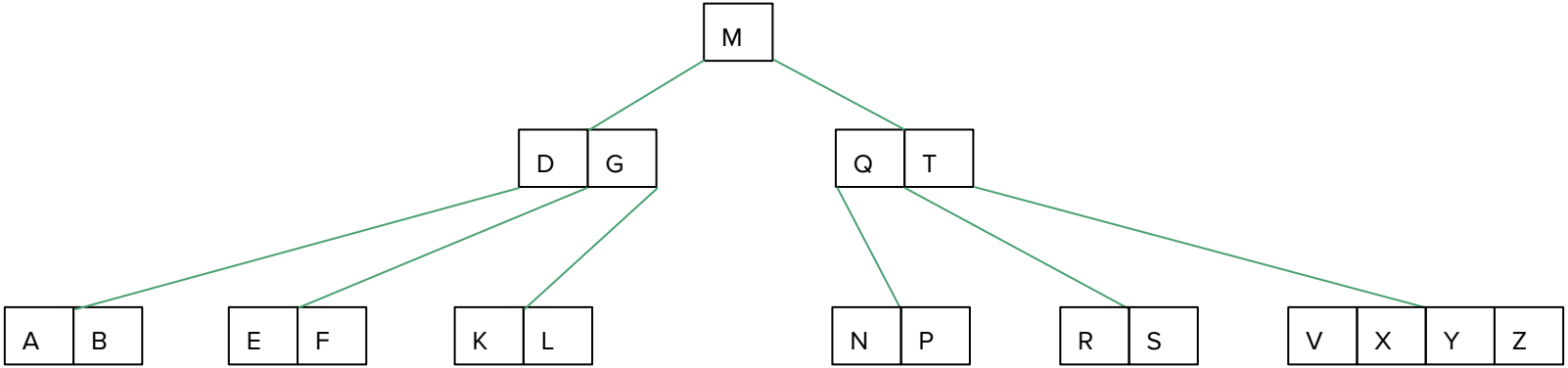
Delete H



Delete H

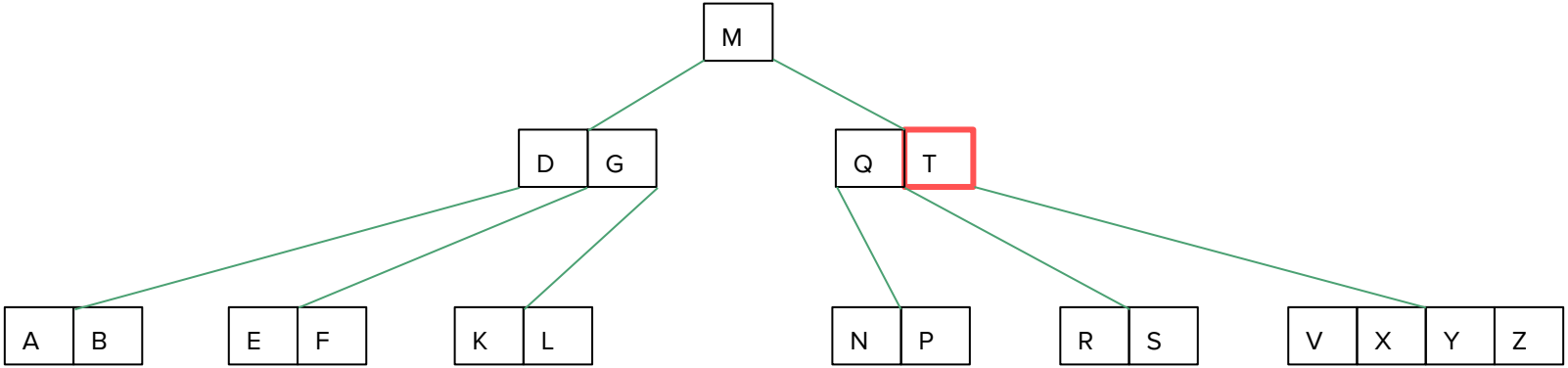


Delete H

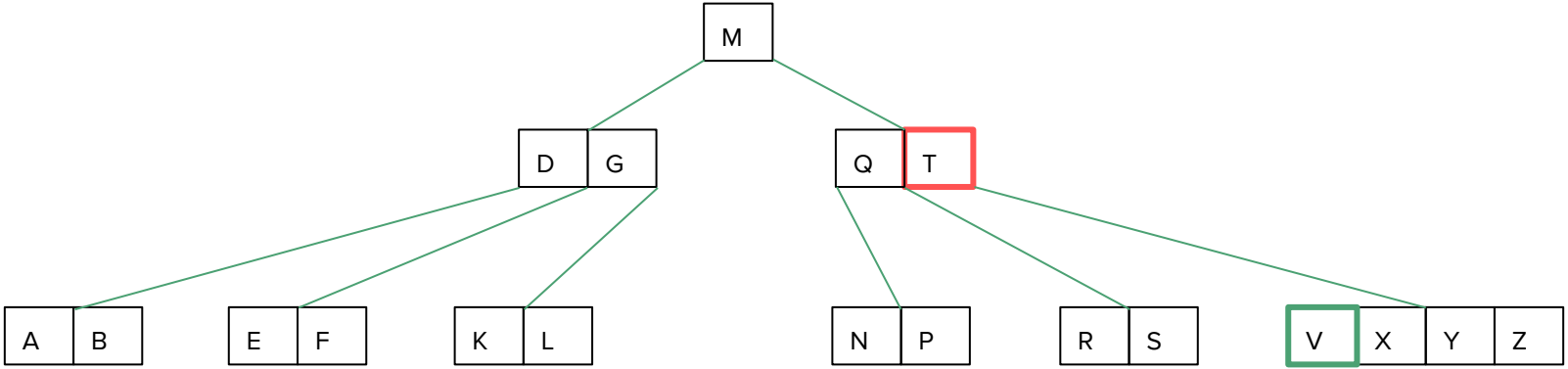


B-Tree Deletion Example 2

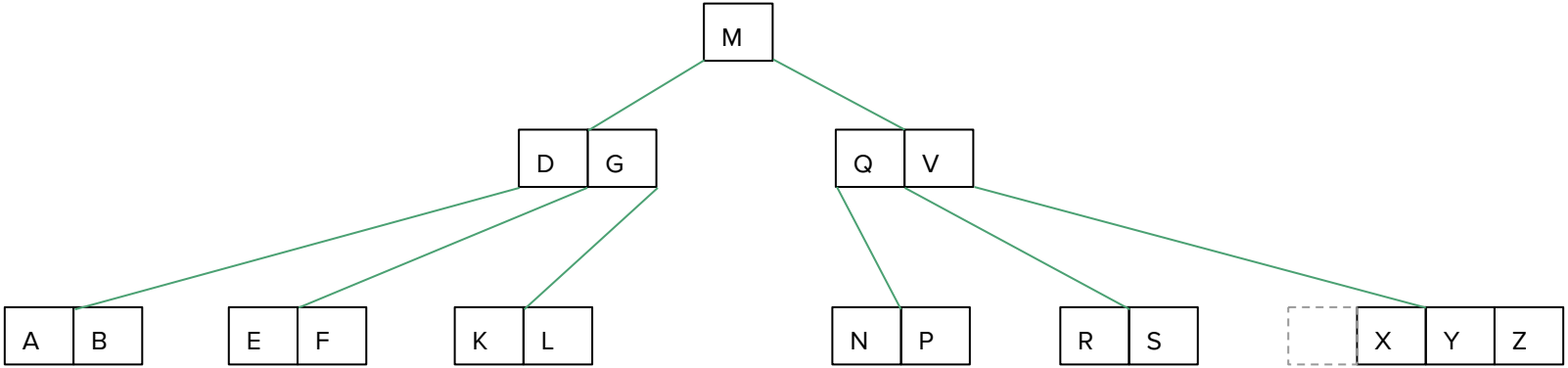
Delete T



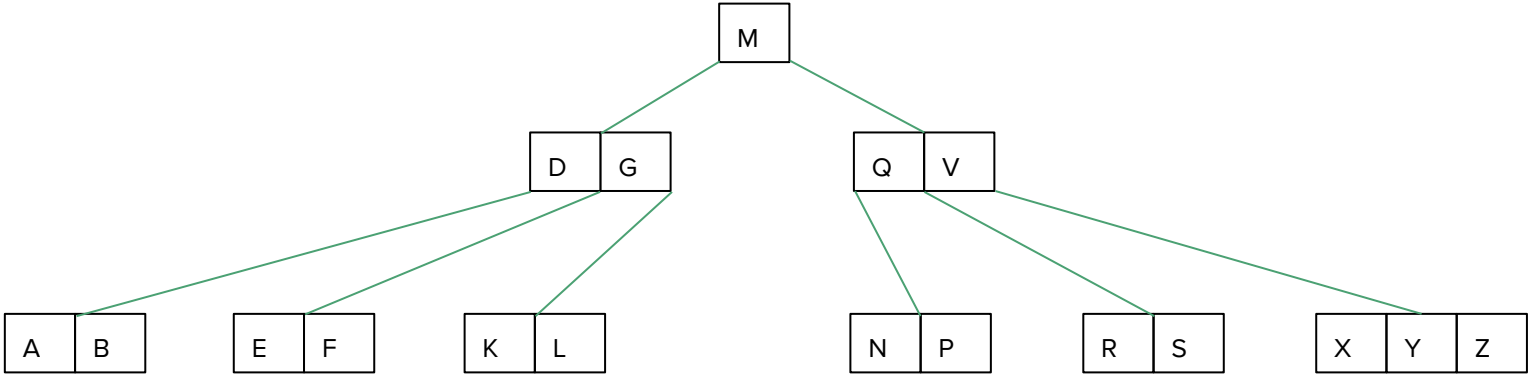
Delete T



Delete T

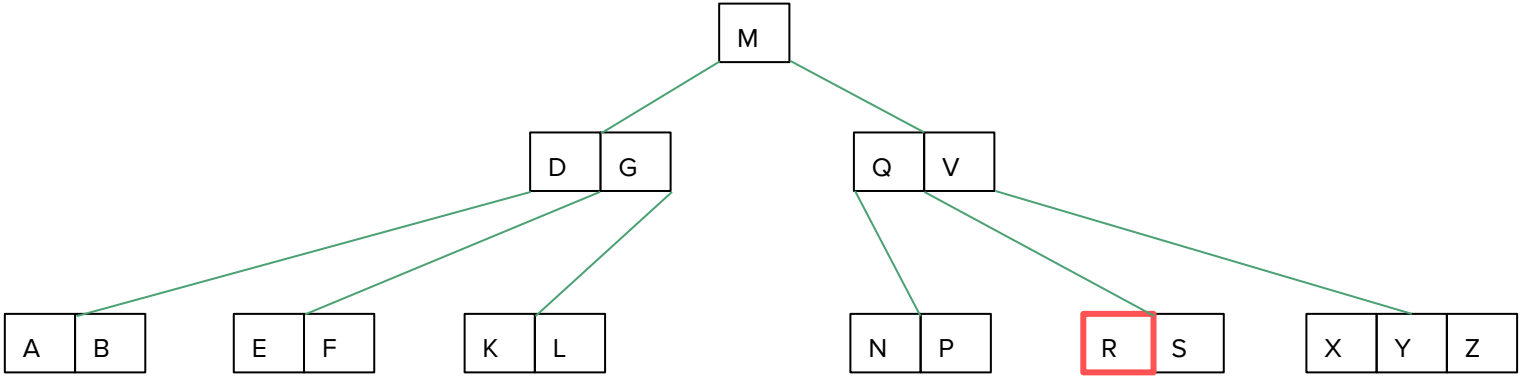


Delete T

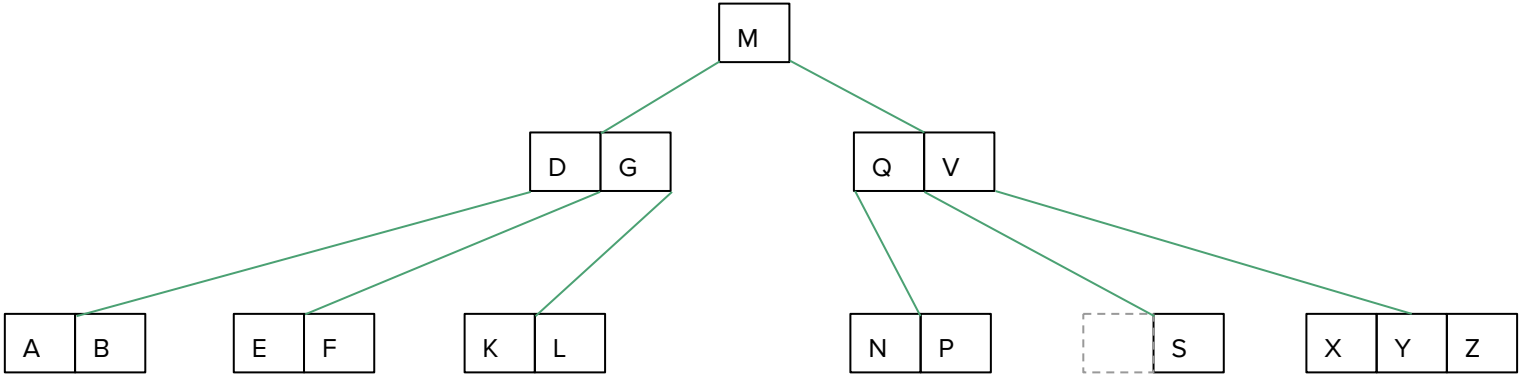


B-Tree Deletion
Example 3

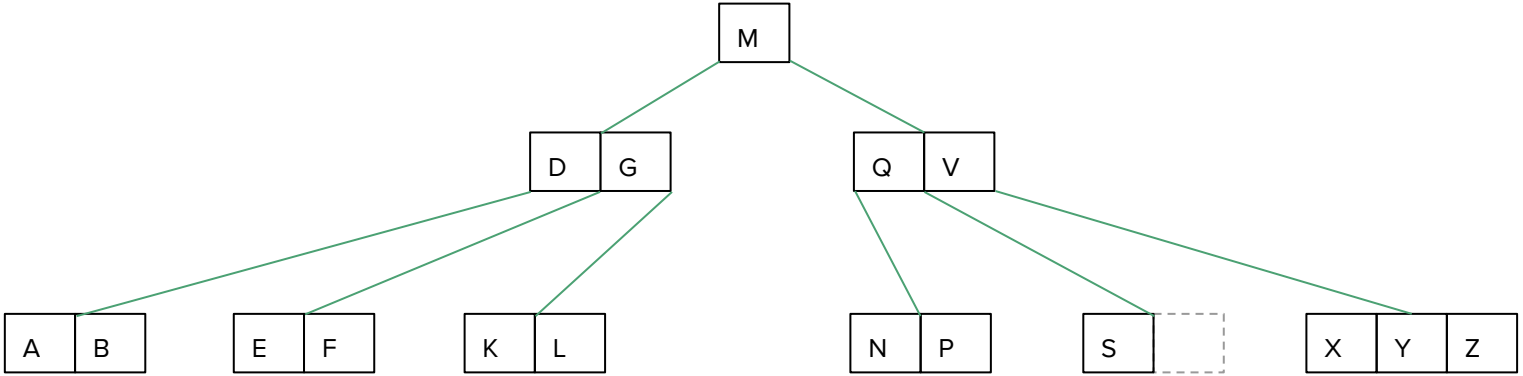
Delete R



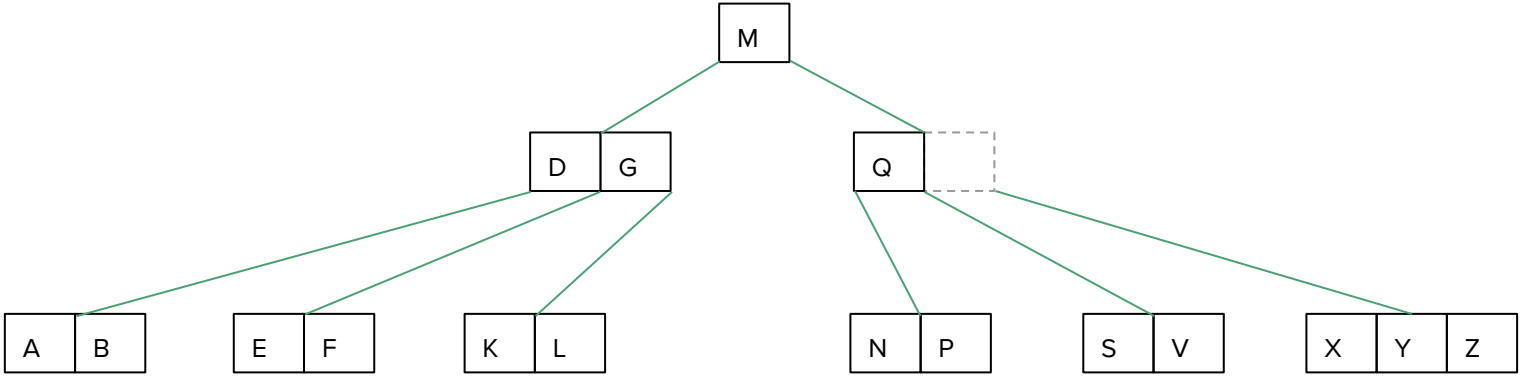
Delete R



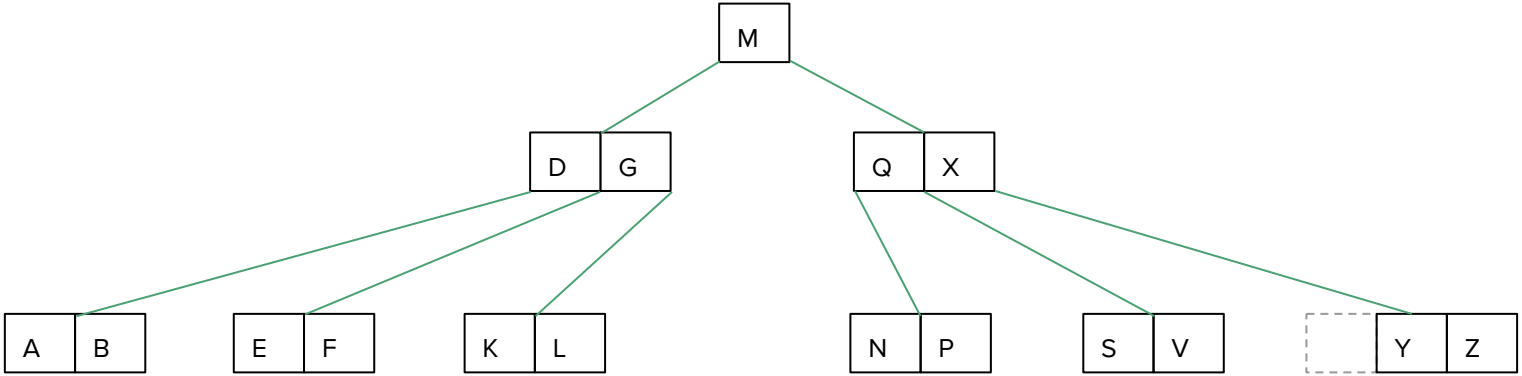
Delete R



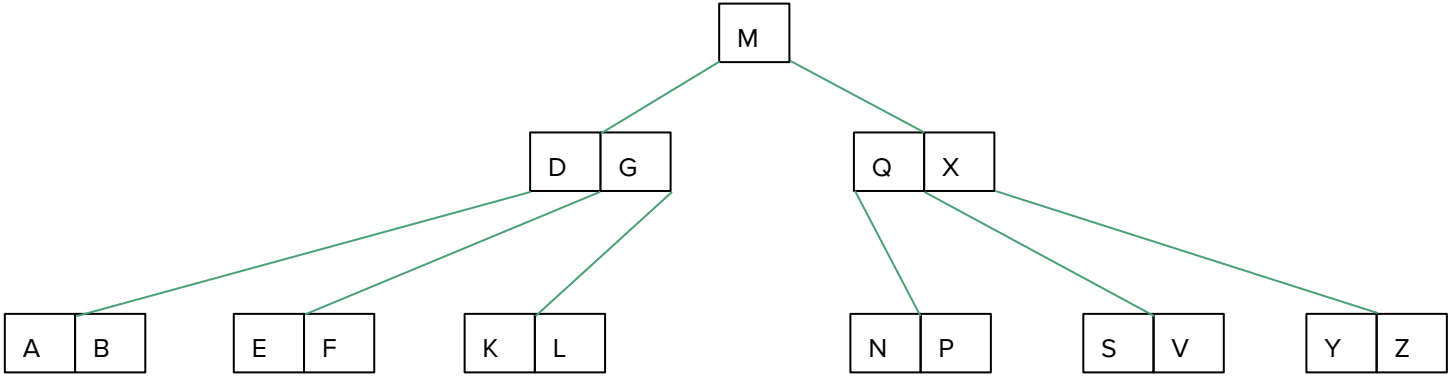
Delete R



Delete R

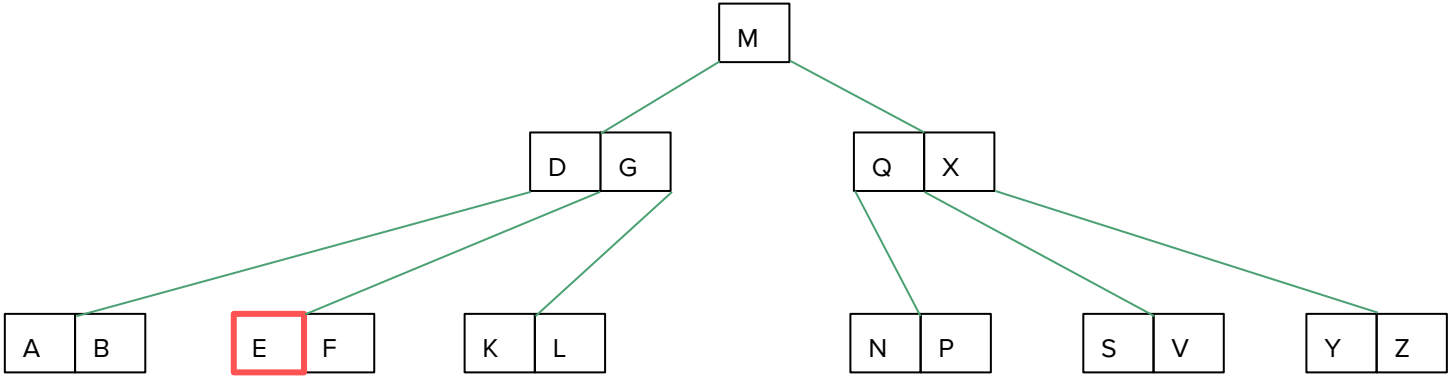


Delete R



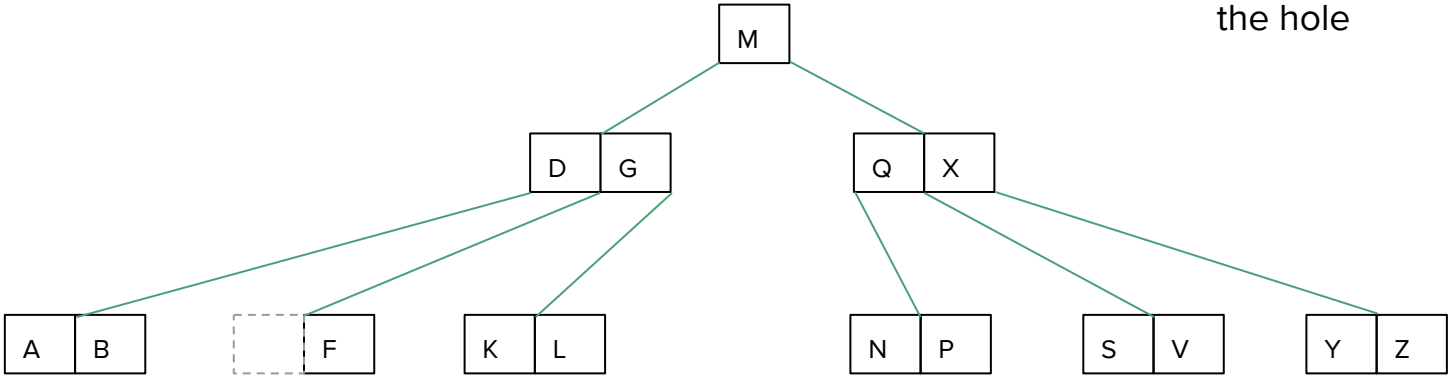
B-Tree Deletion Example 4

Delete E using In-Order Predecessor

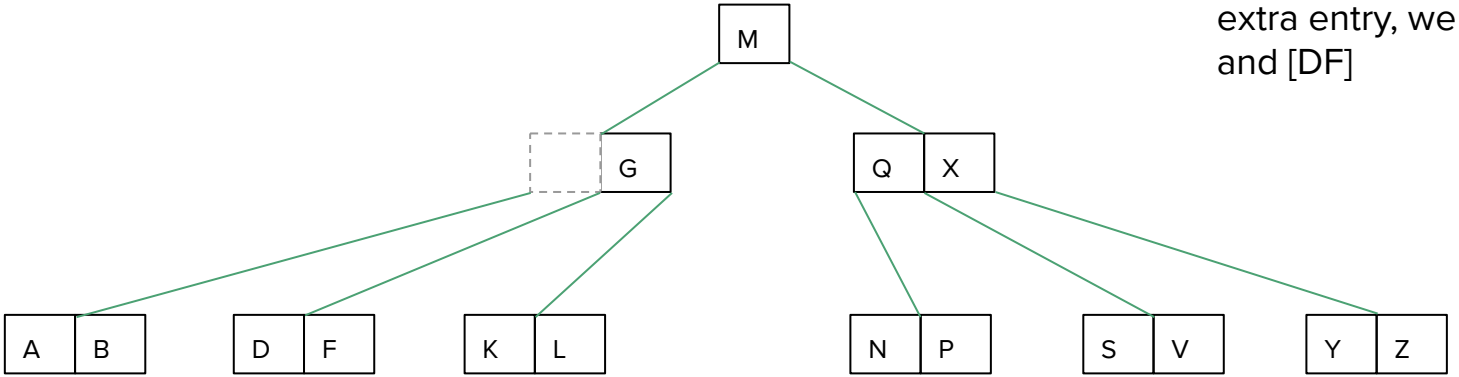


Delete E using In-Order Predecessor

[D] is the predecessor of the hole

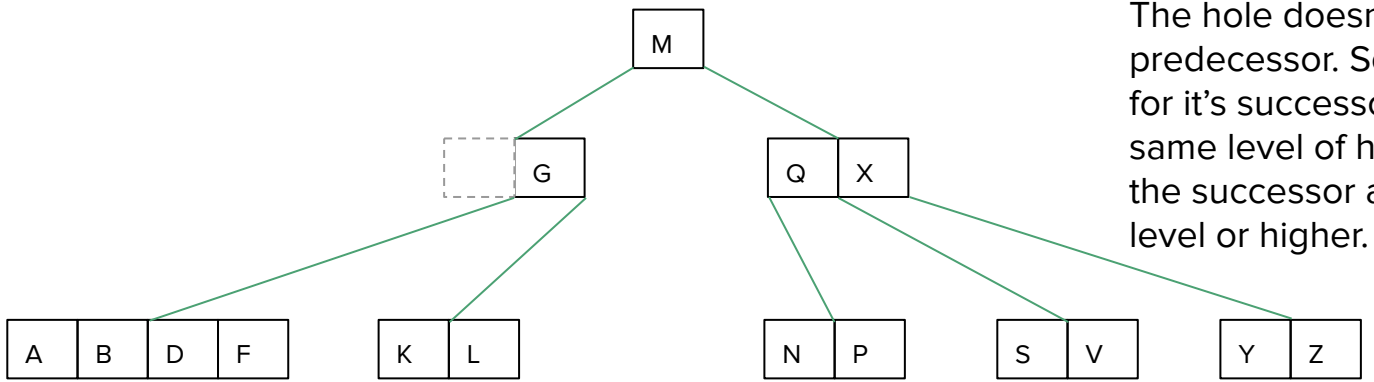


Delete E using In-Order Predecessor



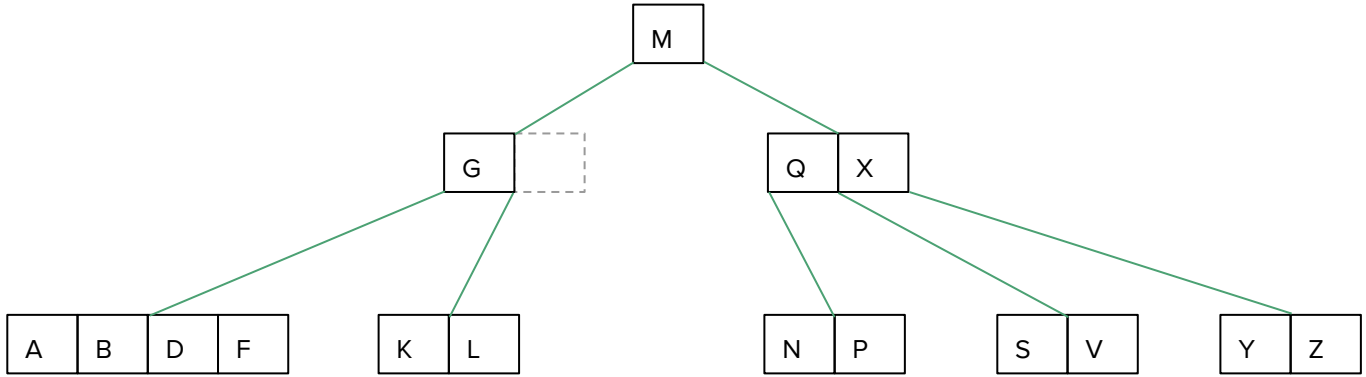
Since [AB] doesn't have an extra entry, we merge [AB] and [DF]

Delete E using In-Order Predecessor



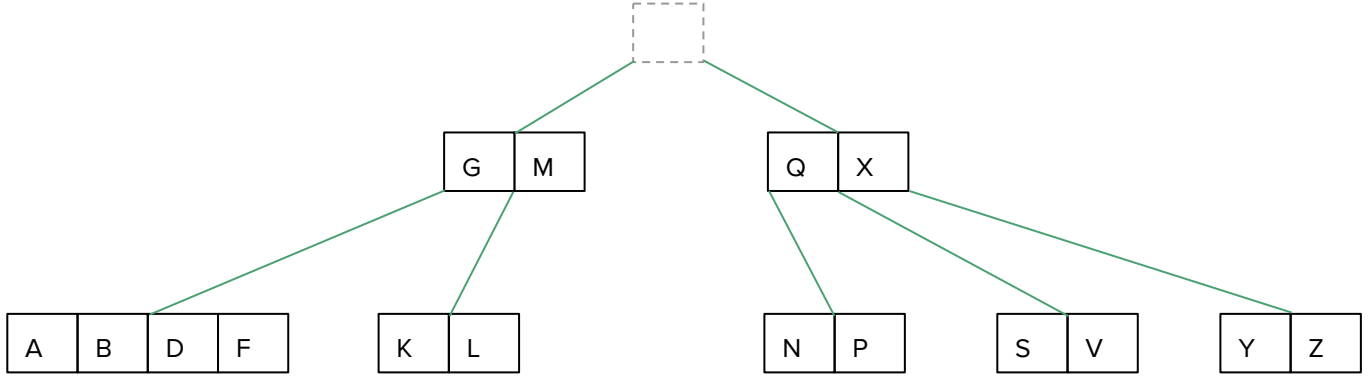
After merging, we look at the same level or higher. The hole doesn't have a predecessor. So, we look for it's successor at the same level of higher. [G] is the successor at the same level or higher.

Delete E using In-Order Predecessor



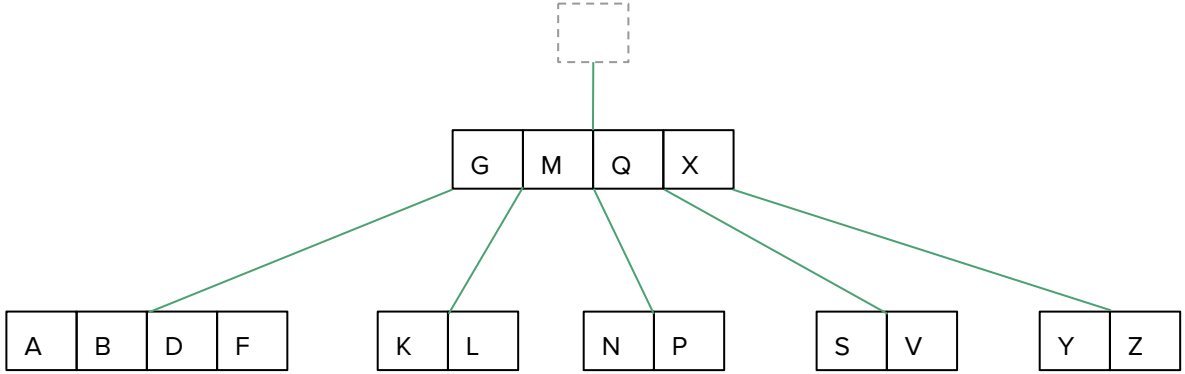
Now, the restriction has gone. You can now fill the hole with a key that's lower, at the same level or higher. [L] is the predecessor of the hole. But since [L] doesn't have a key to spare, and the hole has just one child so we can't even merge the children. Since we can't use the predecessor method, we now look for the hole's successor, [M]

Delete E using In-Order Predecessor

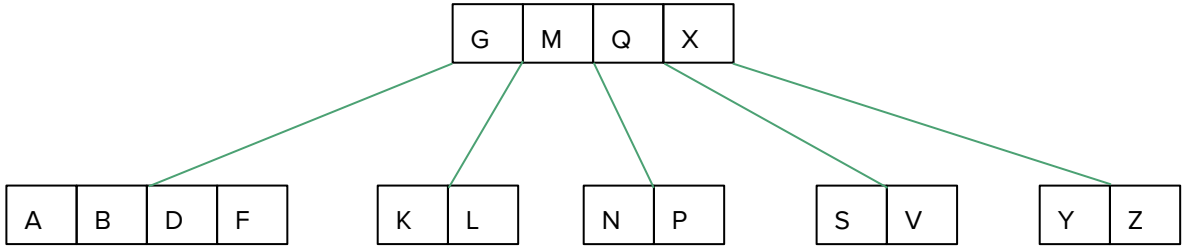


[GM] is the predecessor of the hole. Since [GM] doesn't have an extra entry, we look at its successor, [QX]. Since [QX] also doesn't have a key to spare, we merge [GM] and [QX] and accommodate the children.

Delete E using In-Order Predecessor

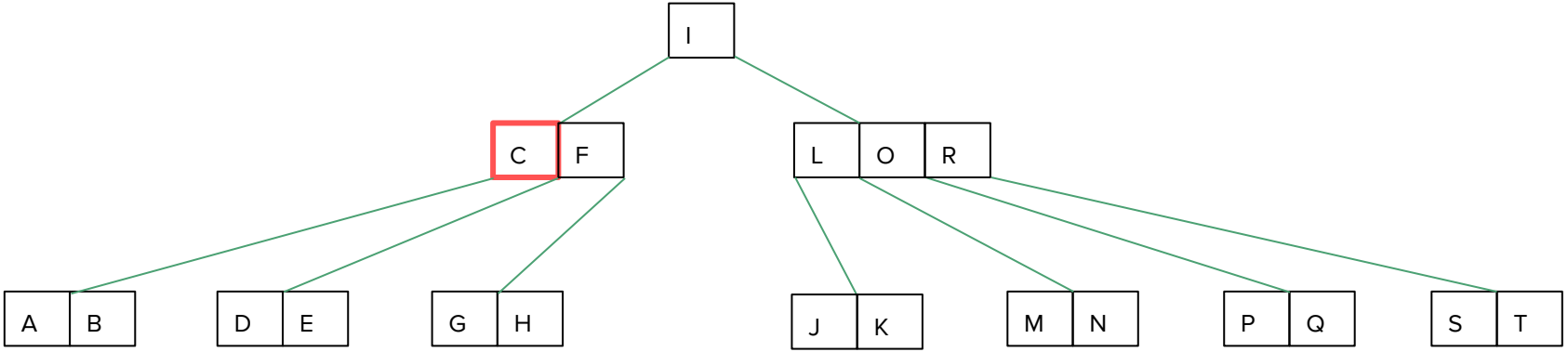


Delete E using In-Order Predecessor

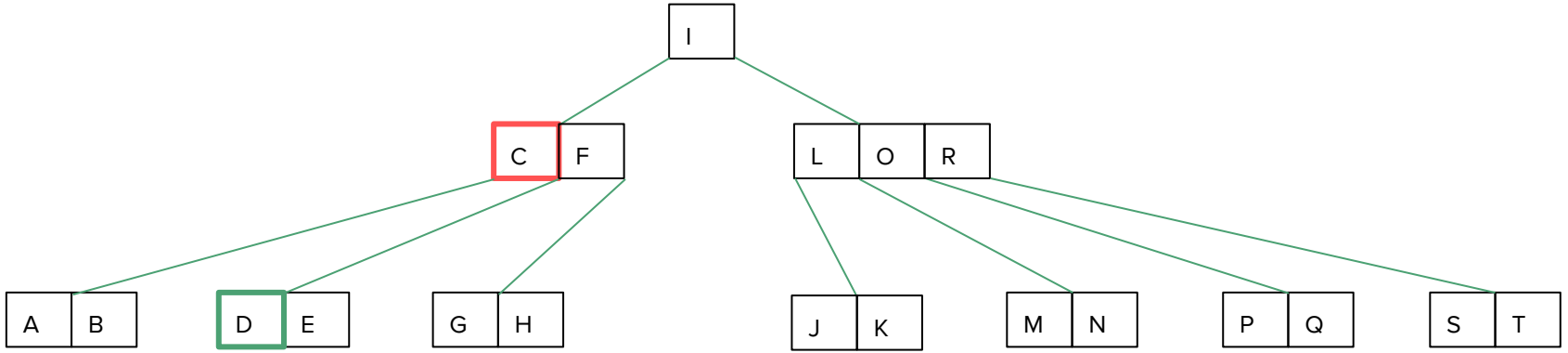


B-Tree Deletion Example 5

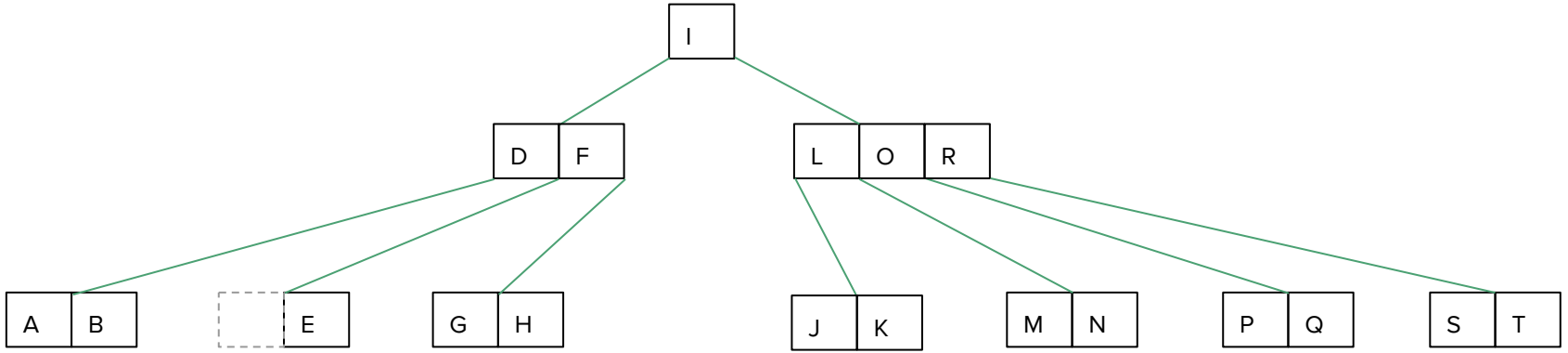
Delete C



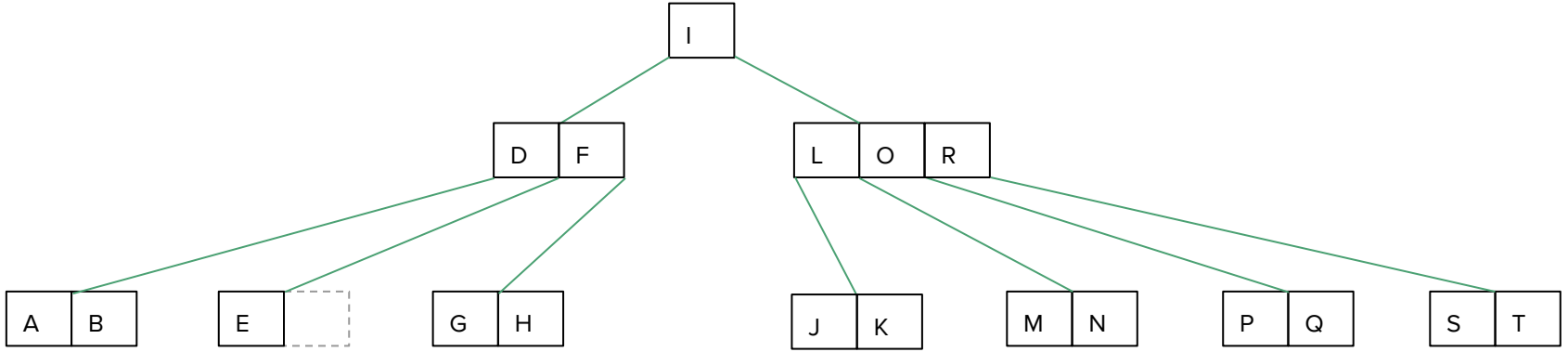
Delete C



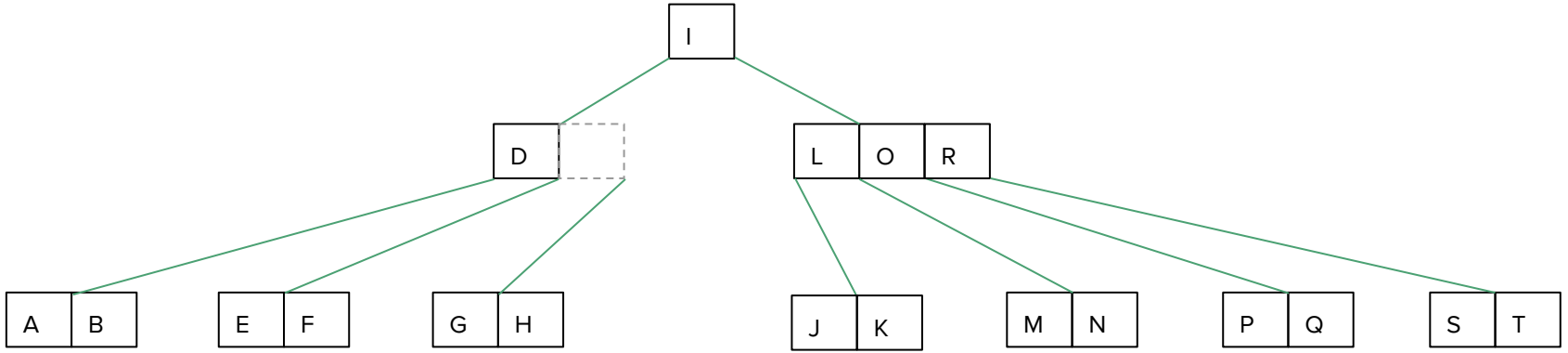
Delete C



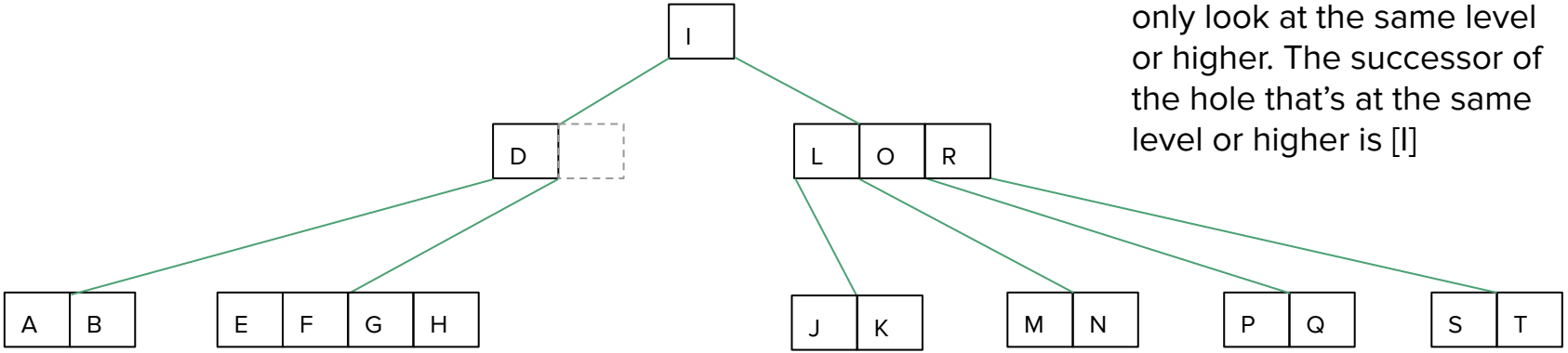
Delete C



Delete C

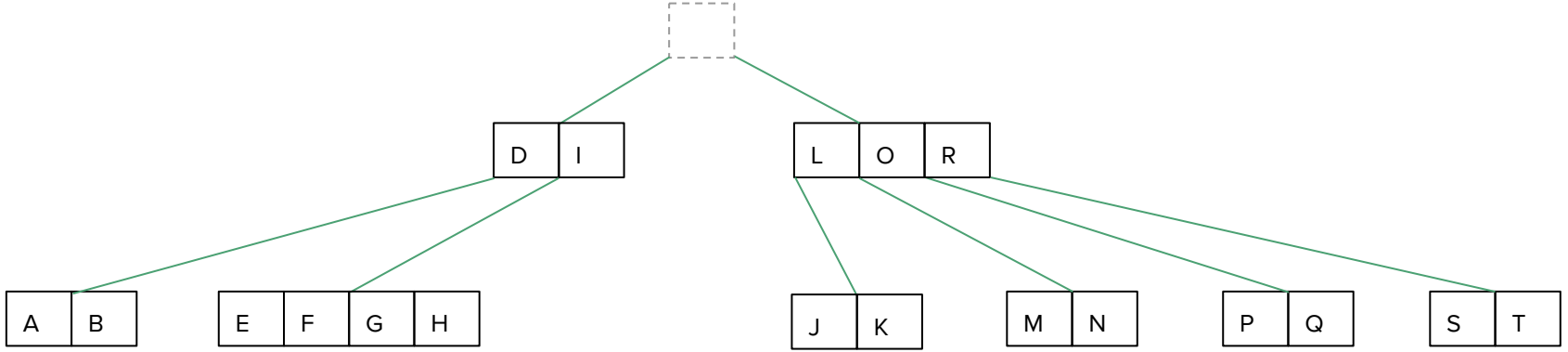


Delete C



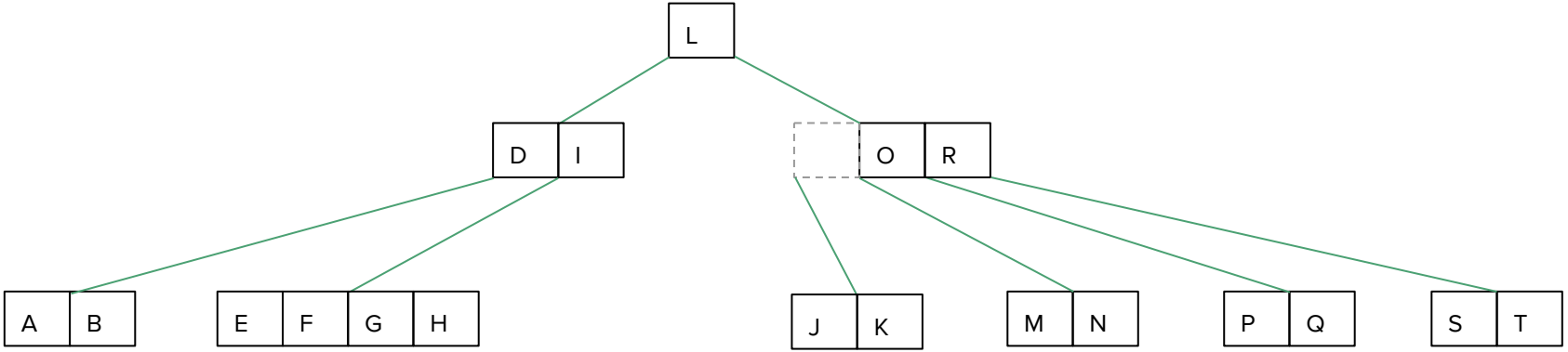
Since we merged, we can only look at the same level or higher. The successor of the hole that's at the same level or higher is [I]

Delete C

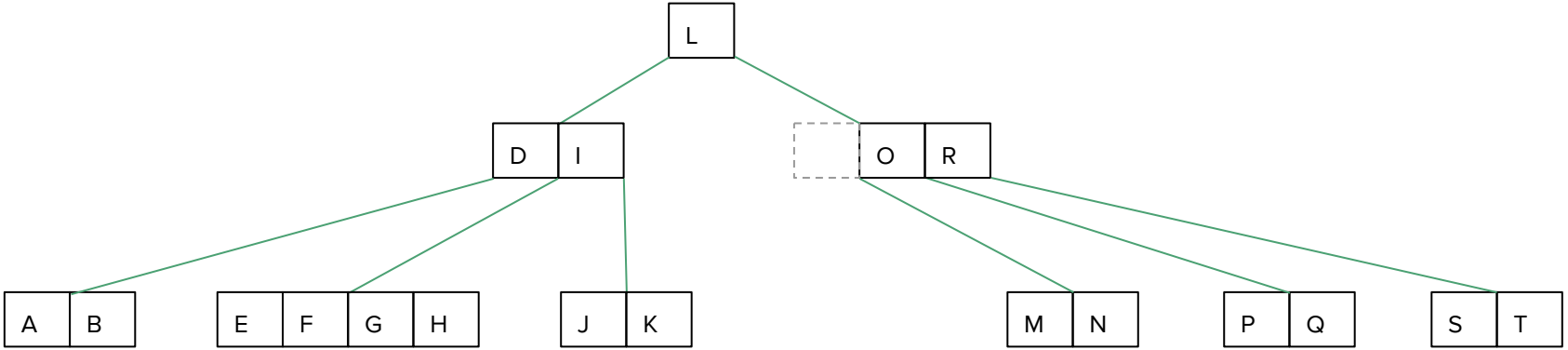


Delete C

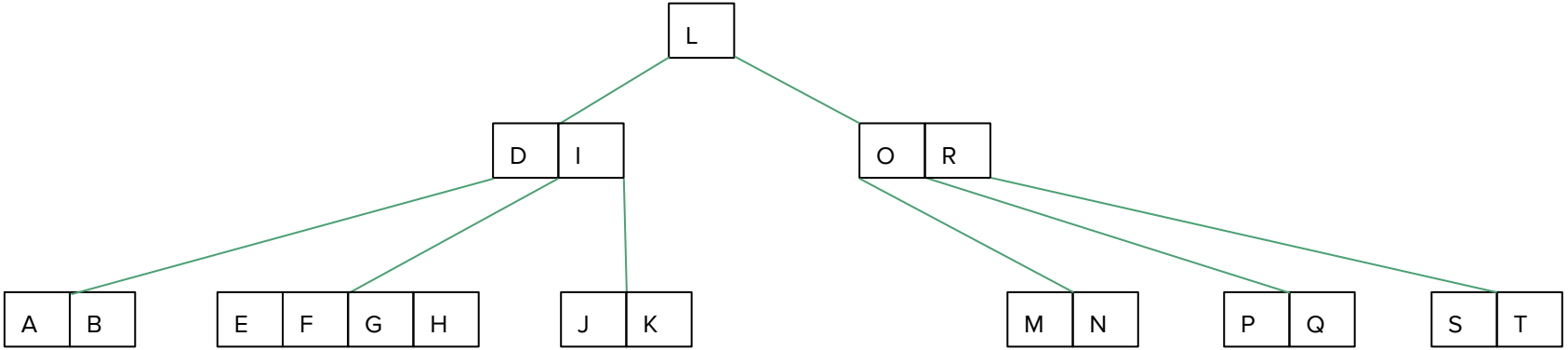
Why adoption?



Delete C

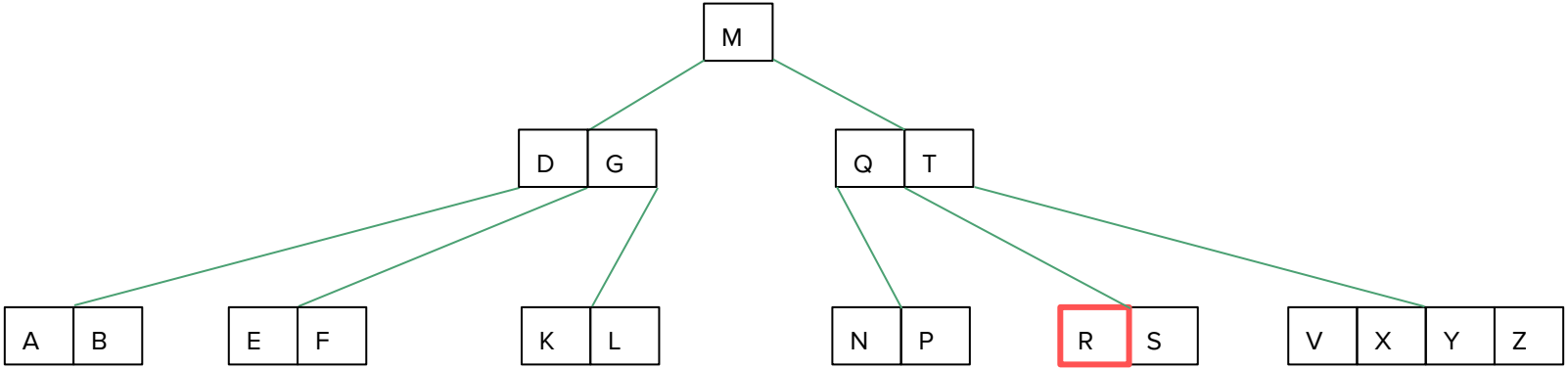


Delete C

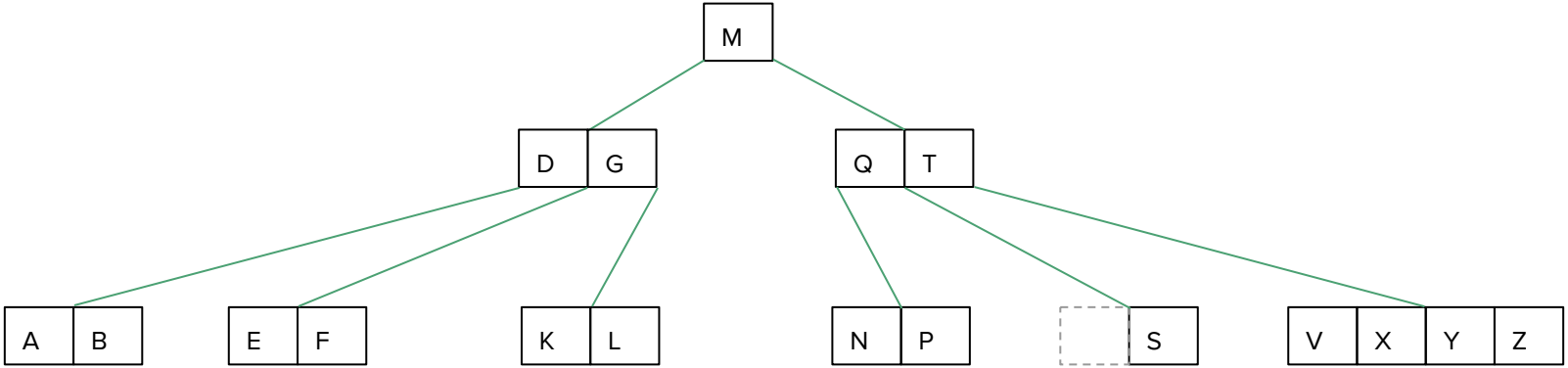


B-Tree Deletion
Example 6

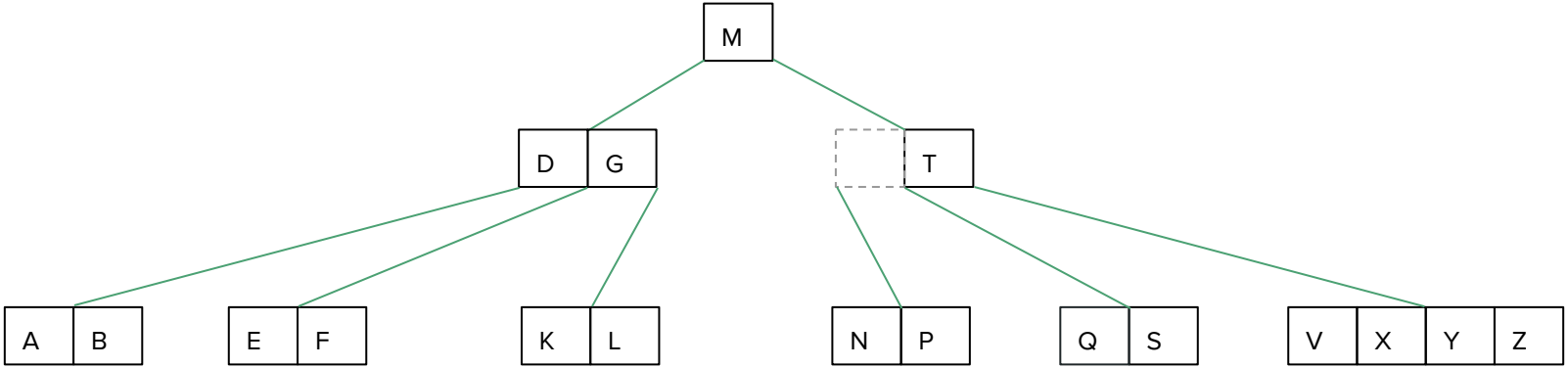
Delete R using In-Order Predecessor



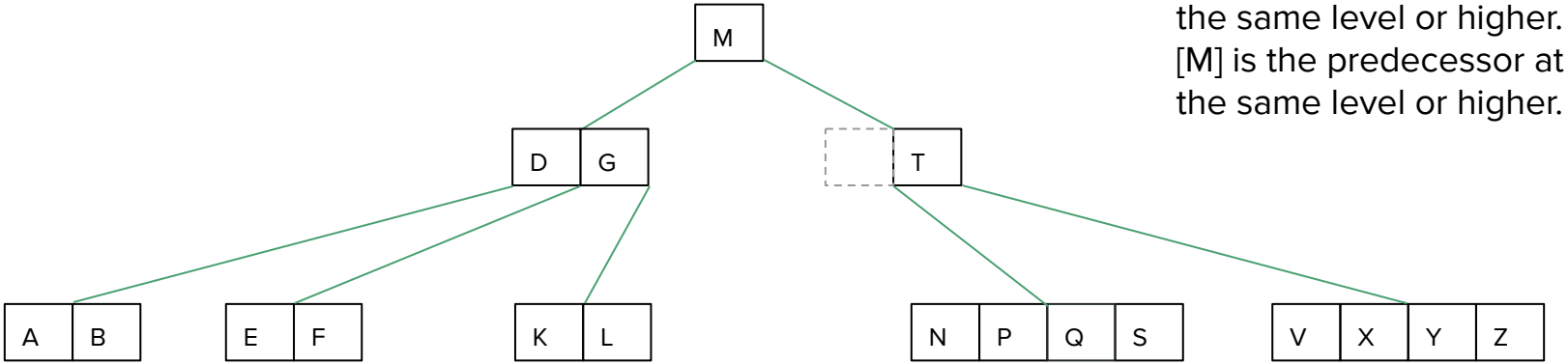
Delete R using In-Order Predecessor



Delete R using In-Order Predecessor

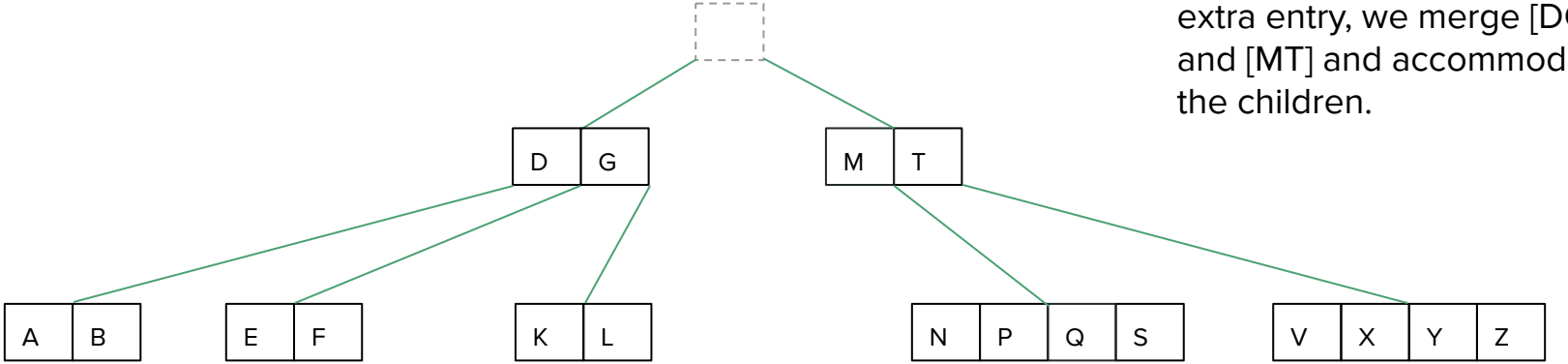


Delete R using In-Order Predecessor



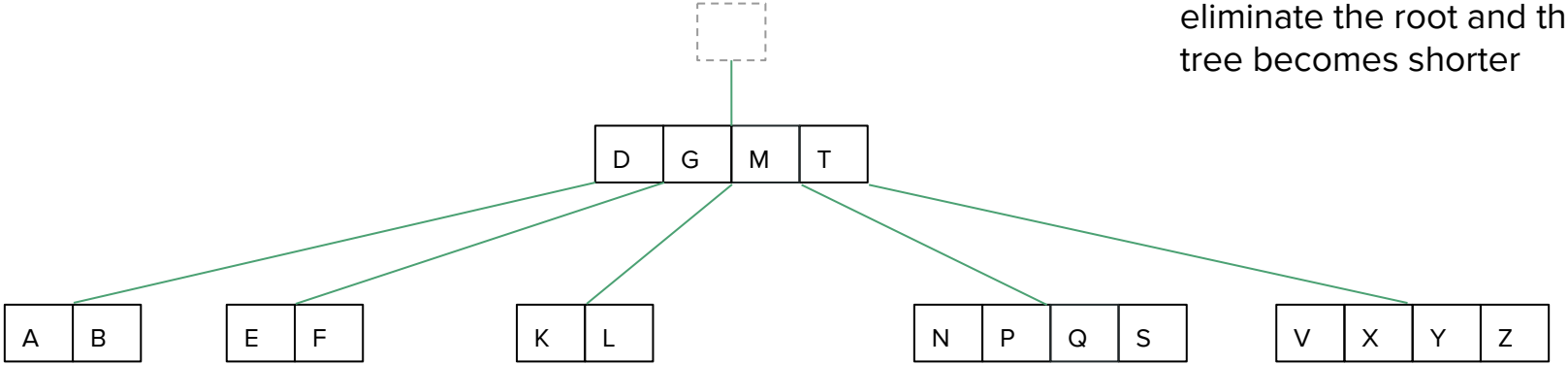
After the merge, we look at the same level or higher. [M] is the predecessor at the same level or higher.

Delete R using In-Order Predecessor



Since [DG] doesn't have an extra entry, we merge [DG] and [MT] and accommodate the children.

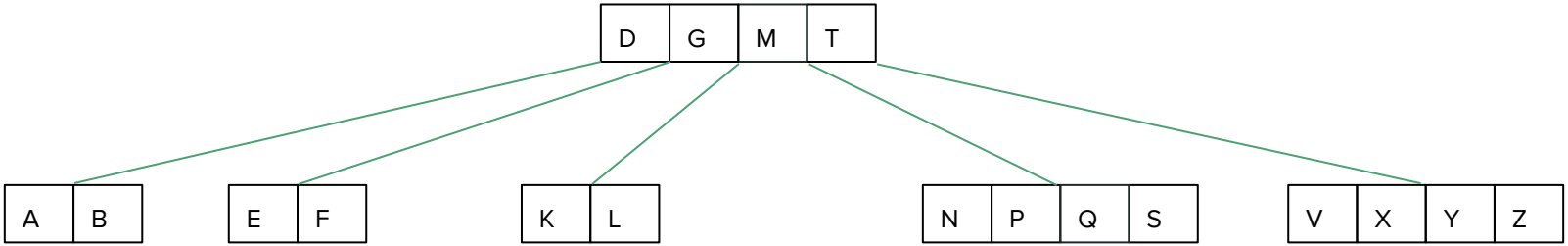
Delete R using In-Order Predecessor



Since the root is a hole, we eliminate the root and the tree becomes shorter

Delete R using In-Order Predecessor

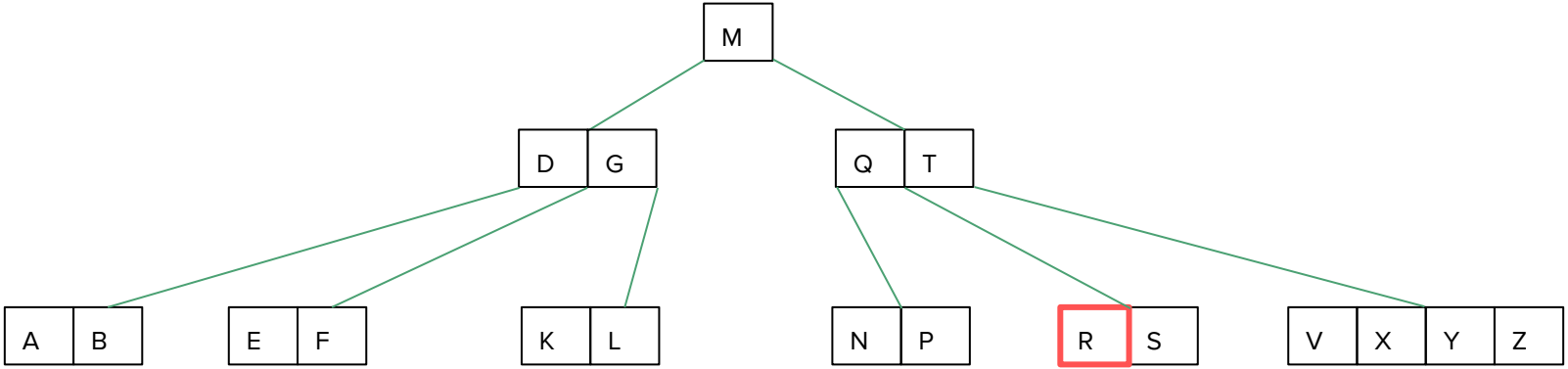
Since the root is a hole, we eliminate the root and the tree becomes shorter



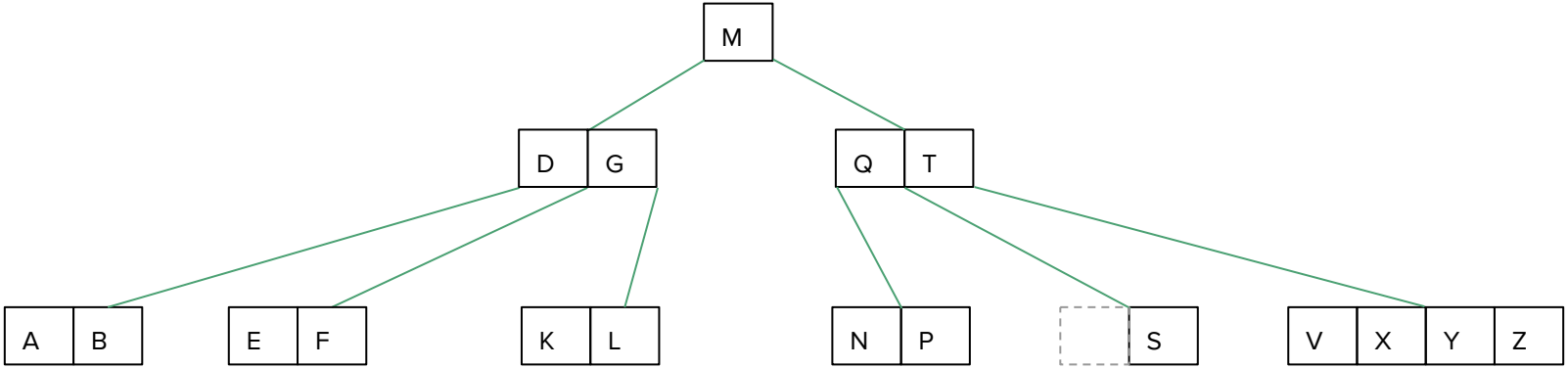
B-Tree Deletion Example 7

(Same as Example 6 but In-Order Successor)

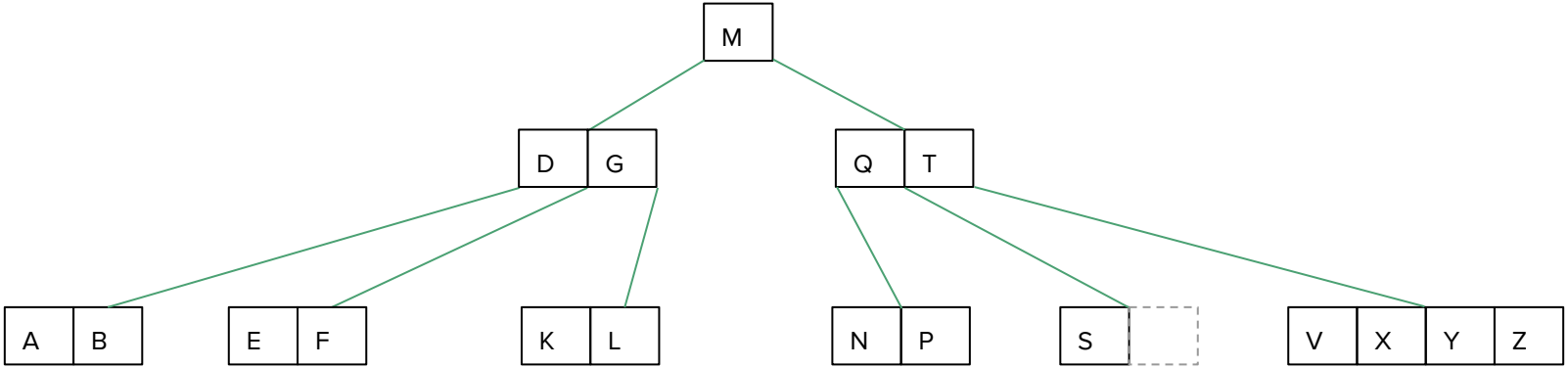
Delete R using In-Order Successor



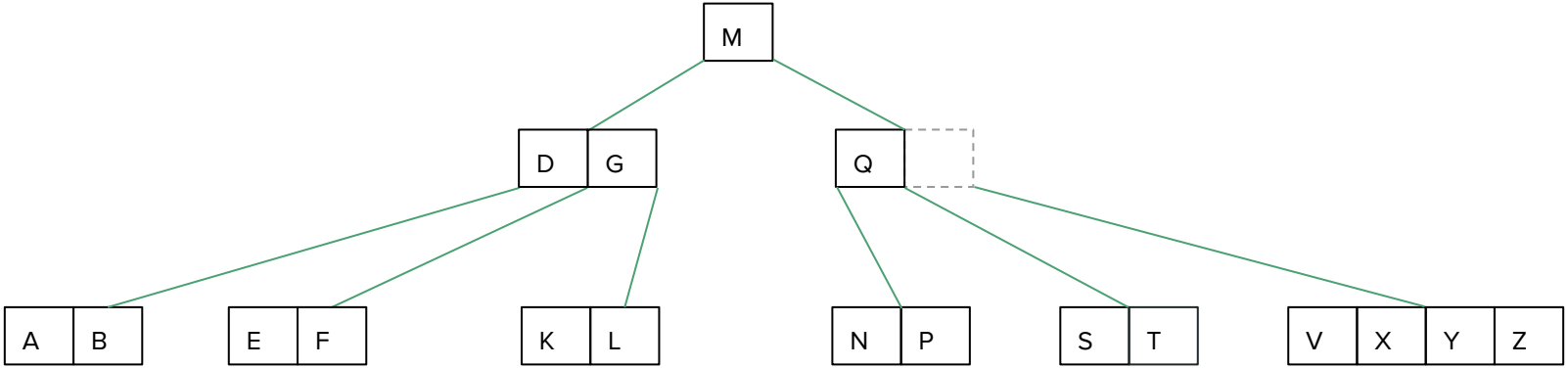
Delete R using In-Order Successor



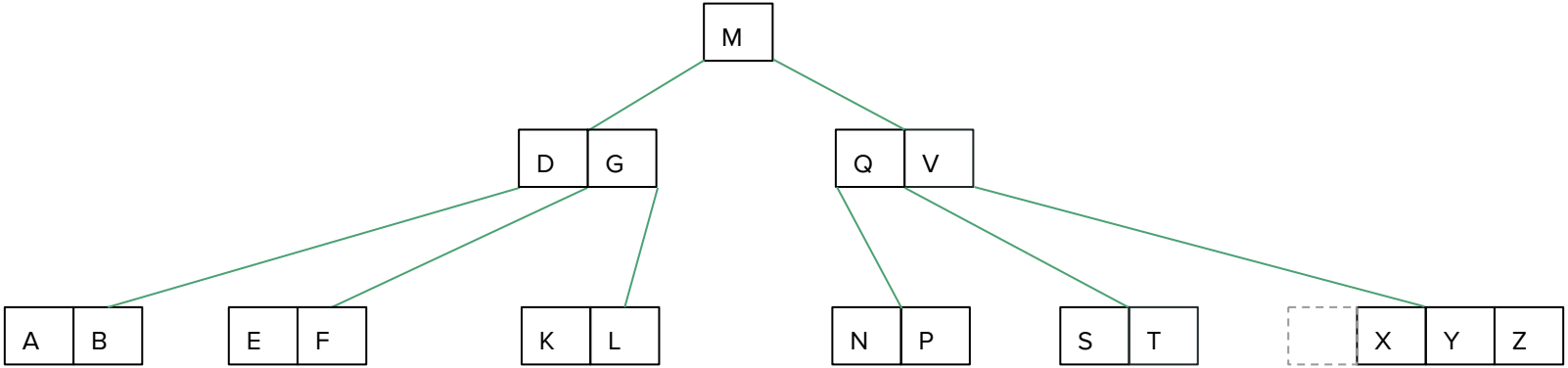
Delete R using In-Order Successor



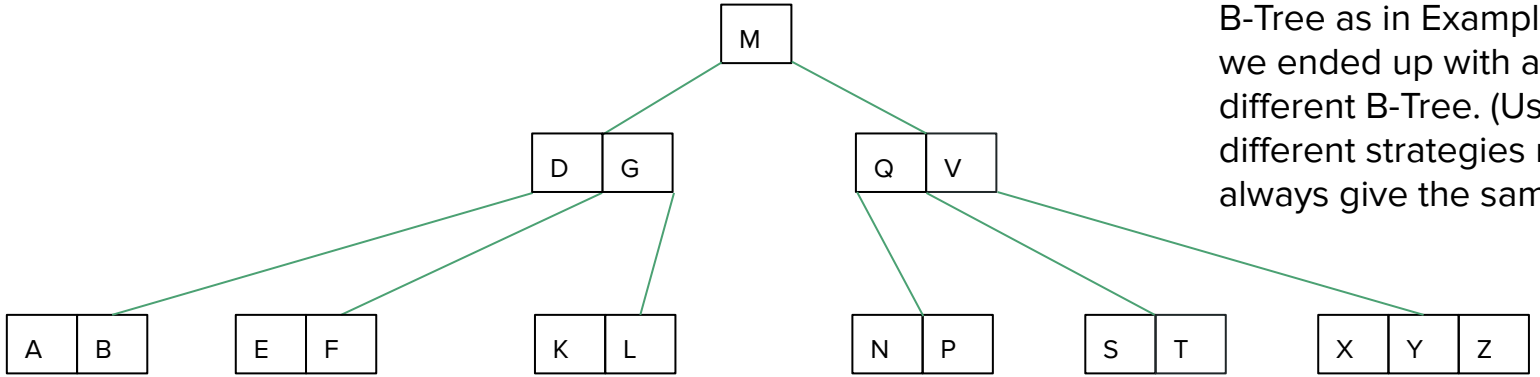
Delete R using In-Order Successor



Delete R using In-Order Successor



Delete R using In-Order Successor

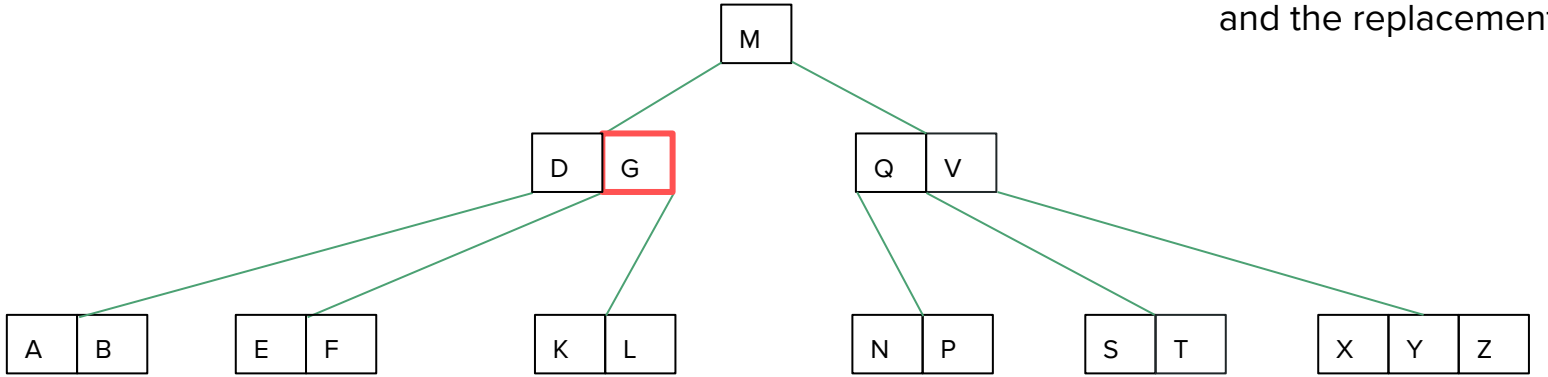


We started with same B-Tree as in Example 6, but we ended up with a different B-Tree. (Using different strategies may not always give the same tree!)

B-Tree Deletion
Example 8

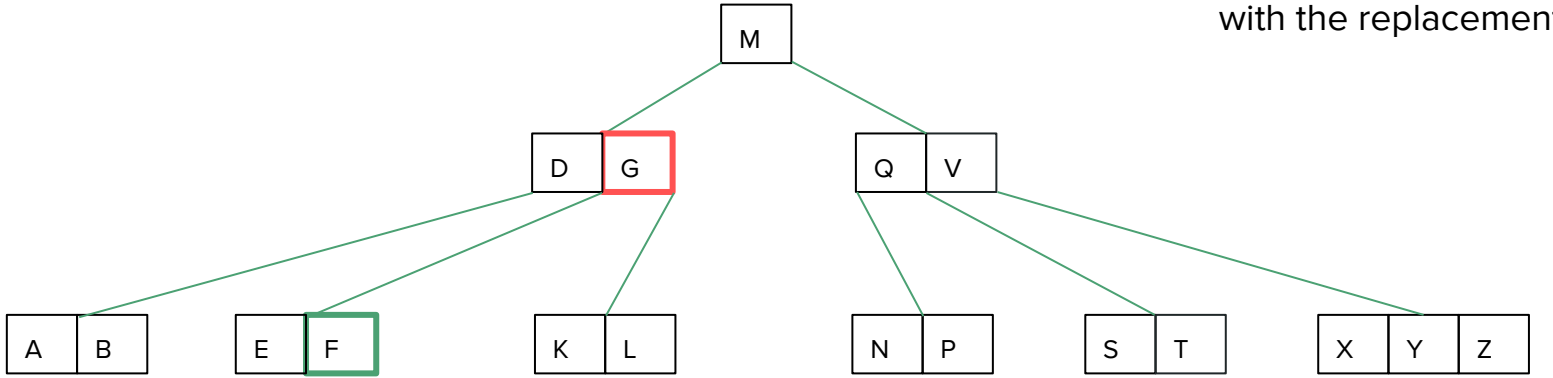
Delete G using In-Order Predecessor

Identify the entry to remove and the replacement entry



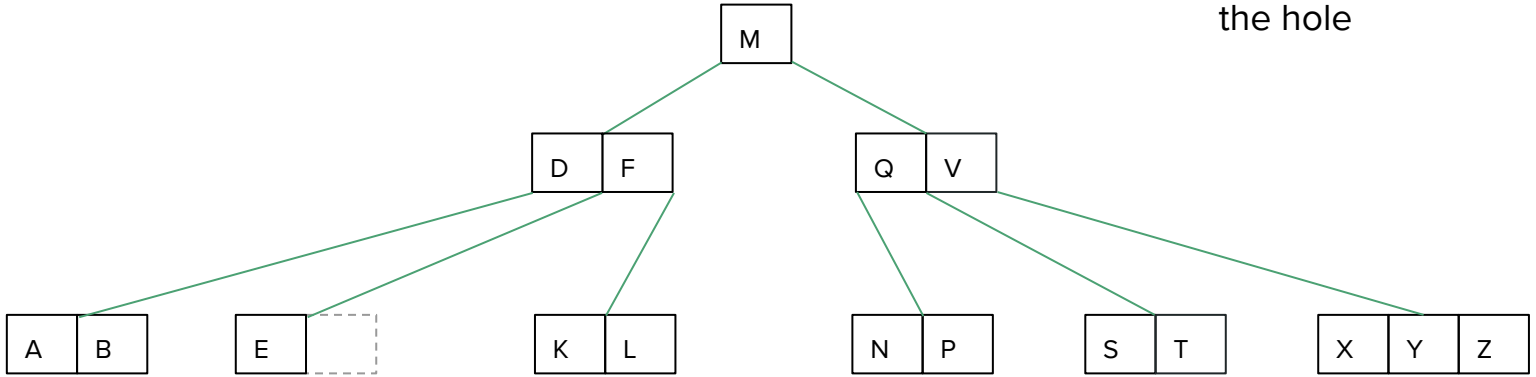
Delete G using In-Order Predecessor

Swap the entry to remove with the replacement entry



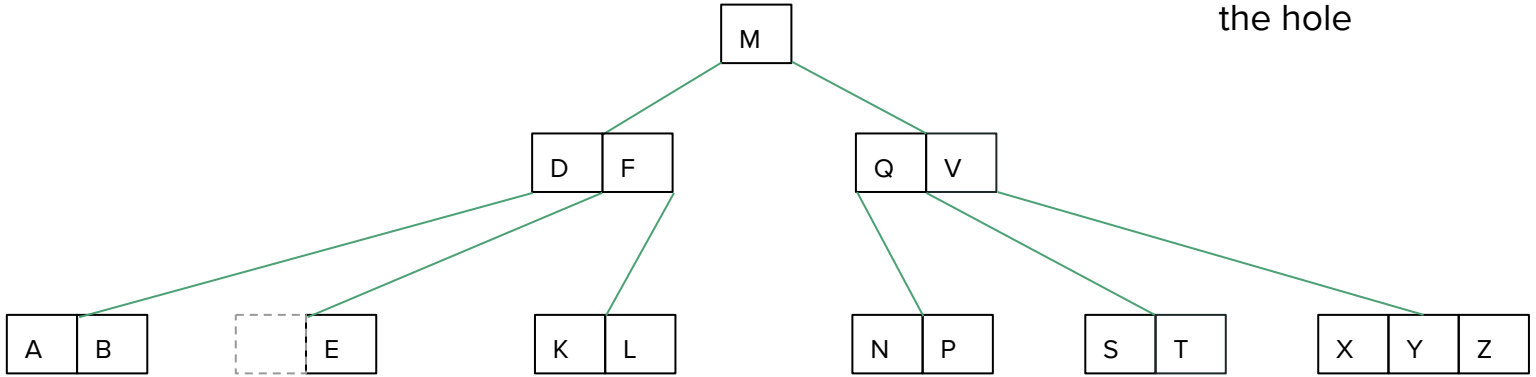
Delete G using In-Order Predecessor

[E] is the predecessor of the hole

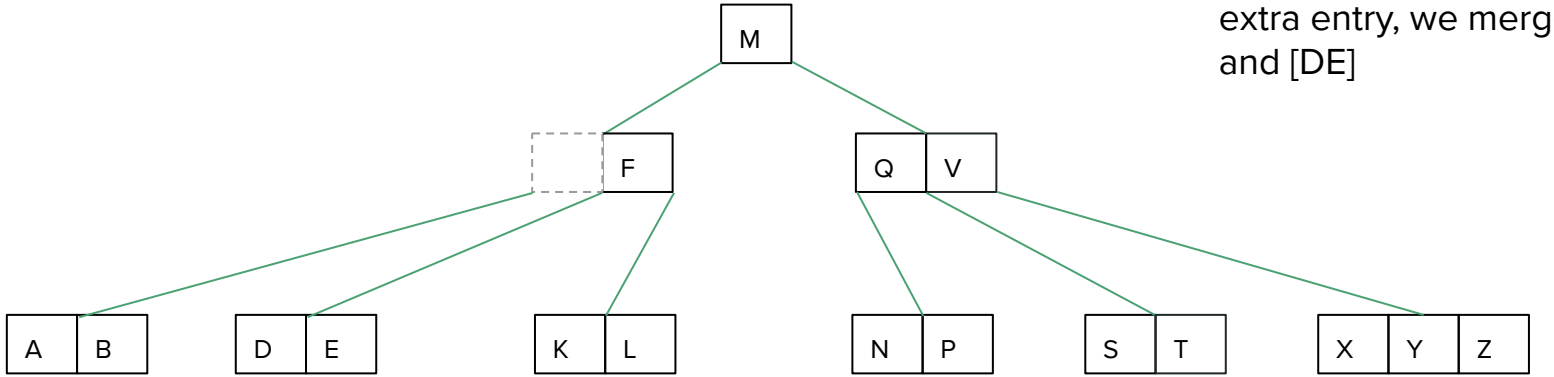


Delete G using In-Order Predecessor

[D] is the predecessor of the hole

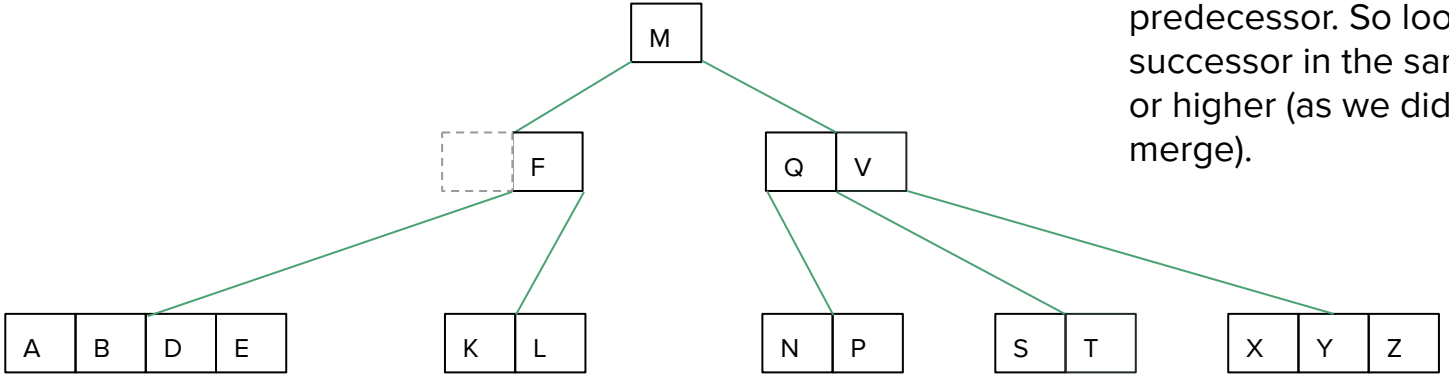


Delete G using In-Order Predecessor



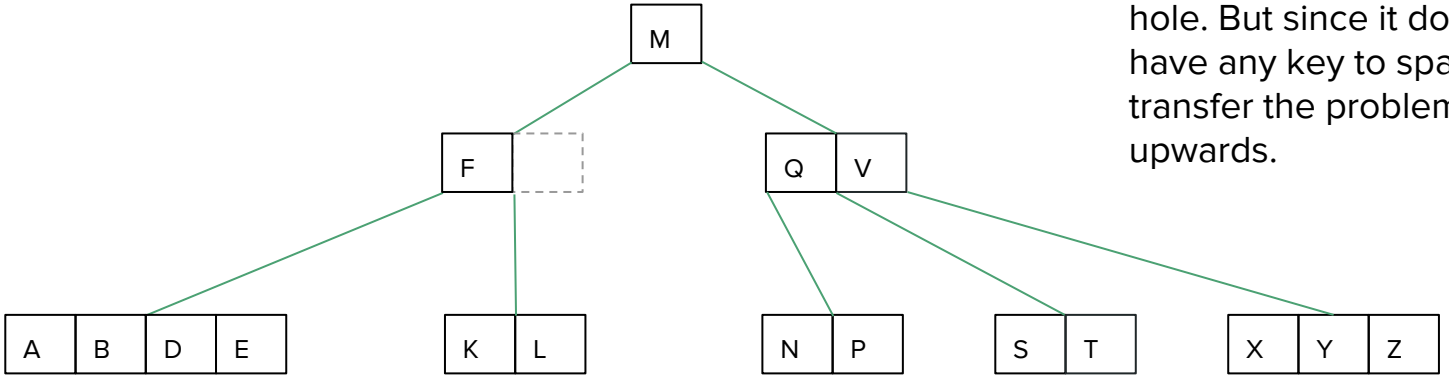
Since [AB] doesn't have an extra entry, we merge [AB] and [DE]

Delete G using In-Order Predecessor



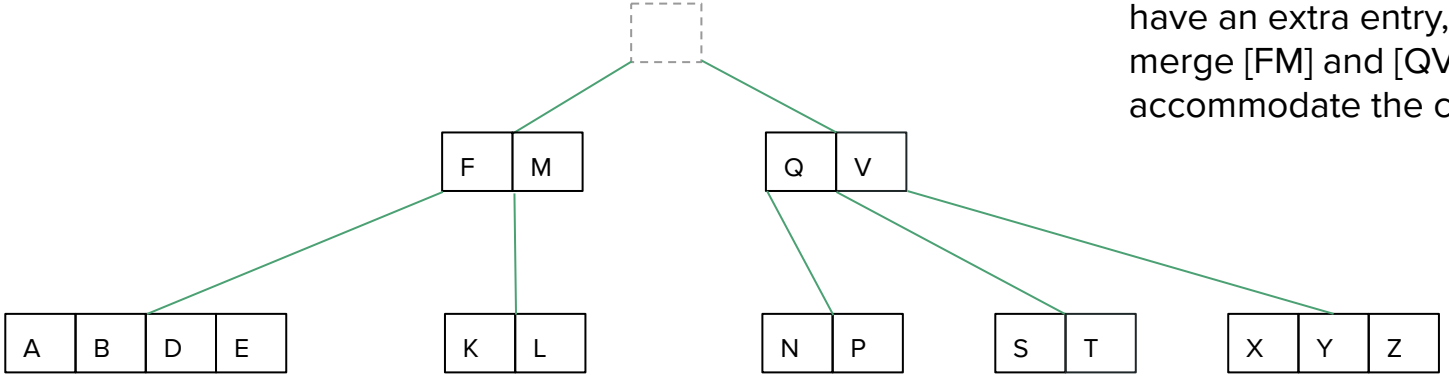
The hole doesn't have a predecessor. So look for its successor in the same level or higher (as we did the merge).

Delete G using In-Order Predecessor



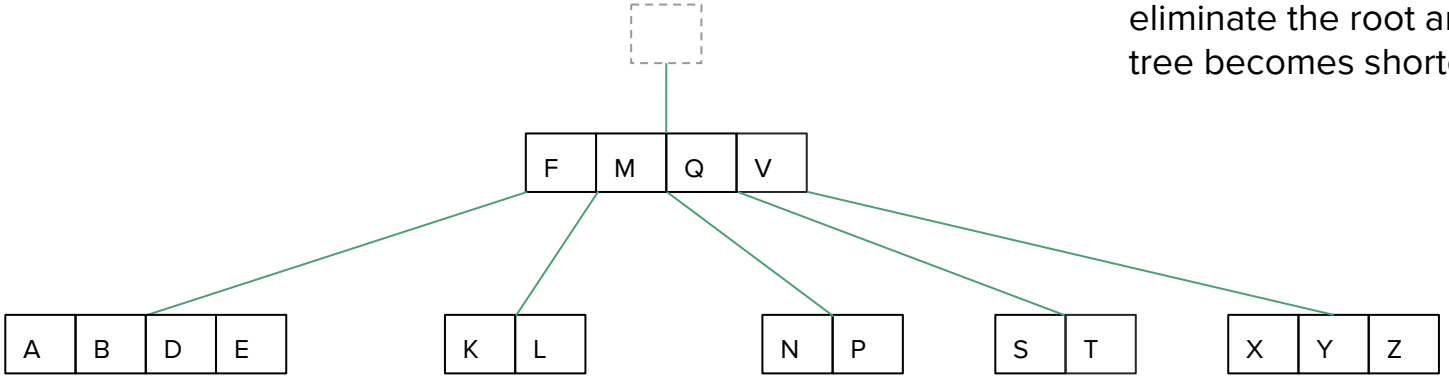
[KL] is the successor of the hole. But since it doesn't have any key to spare, we transfer the problem upwards.

Delete G using In-Order Predecessor



Since [FM] and [QV] don't have an extra entry, we merge [FM] and [QV] and accommodate the children.

Delete G using In-Order Predecessor



Since the root is a hole, we eliminate the root and the tree becomes shorter

Delete G using In-Order Predecessor