Data Structures (15B11CI311)

Odd Semester 2020

3rd Semester , Computer Science and Engineering

Jaypee Institute Of Information Technology (JIIT), Noida

Contents

- Need of B-tree
- B-tree definition
- Insertion in B-tree
- Deletion in B-tree
- Reasons for using B-tree

Need of B-Tree

- When we want to find and retrieve records from a disk file, the time required for a single access is thousands times greater in case of external retrieval.

- So, our goal in external searching is to minimize the no. of disk access since each access takes so long compared to internal computation.

- For external searching, multiway trees are more appropriate .

B-Tree

- A B-Tree is a balanced multiway tree. A node of the tree contains several records or keys of records and pointers to children.

- To reduce the no. of disk access, the following points are applicable:

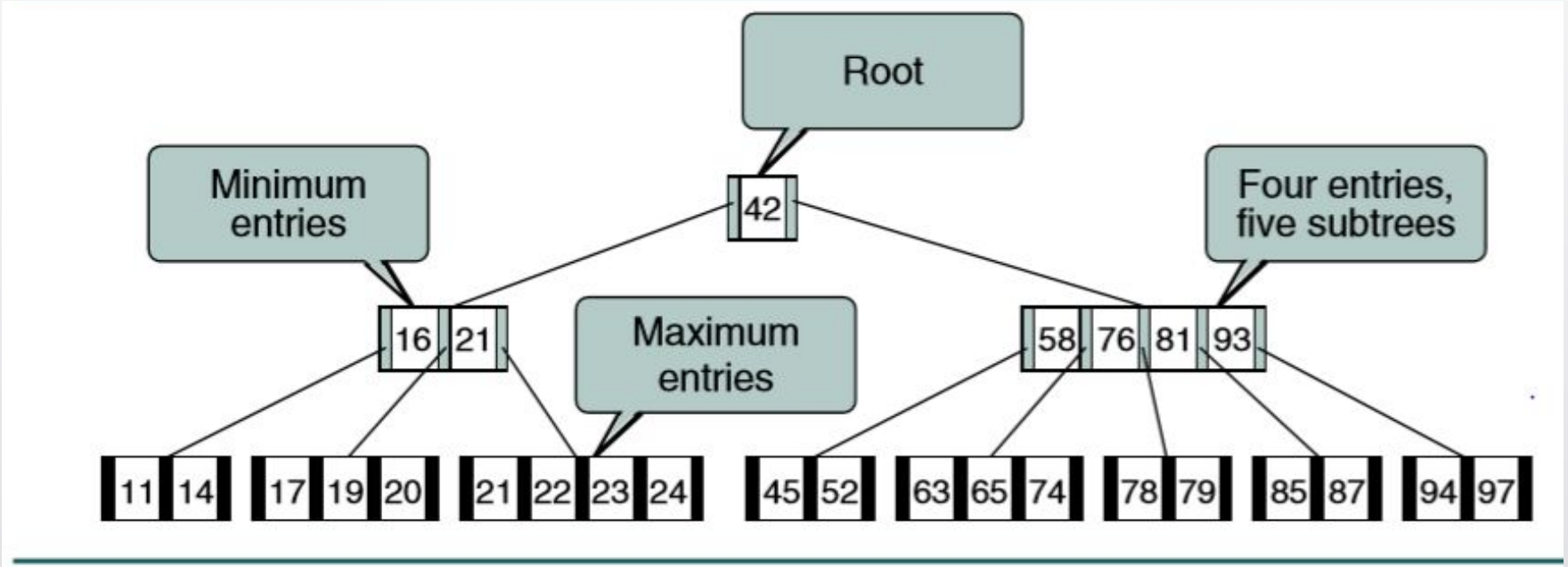
1. Height is kept minimum.
2. All leaves are kept at same level.
3. All nodes other than leaf nodes must have at least minimum no. of children.

B-Tree Definition

A B-Tree of order m is a multiway tree with the following properties:

1. The number of keys in each internal node is one less than the number of its non-empty children.
2. All leaves are on the same level.
3. Each node can have
   Max- children= m
   Max- Keys= m-1
4. All nodes except root node have
   Min-children= $\lceil m/2 \rceil$
   Min- keys= $\lceil m/2 \rceil - 1$
5. The root can be either a leaf node, or it has Min- children=2 and Min-key=1

Example of B-Tree



A B-tree of Order 5

Source: Gilberg, Richard F., and Behrouz A. Forouzan. *Data Structures: A pseudocode approach with C*. Nelson Education, 2004.

Inserting a key into a B-tree

Binary search trees grow at their leaves, but the B-trees grow at their root. The algorithm of insertion is as follows:

1. Attempt to insert the new key into a leaf.
2. If the leaf node is not full, then the new key is added to it and the insertion is finished.
3. If the leaf node is full, then it splits into two nodes on the same level, except that the median key is sent up the tree to be inserted into the parent node.

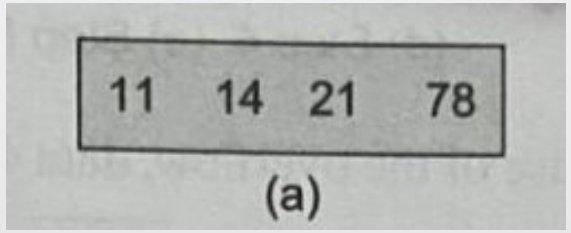
Inserting a key into a B-tree (contd..)

4. If this would result in the parent becoming too big, split the parent into two, promoting the middle key.
5. This strategy might have to be repeated all the way to the top.
6. If necessary, the root can be split into two and the middle key is promoted to a new root and the resulting tree will be one level higher.

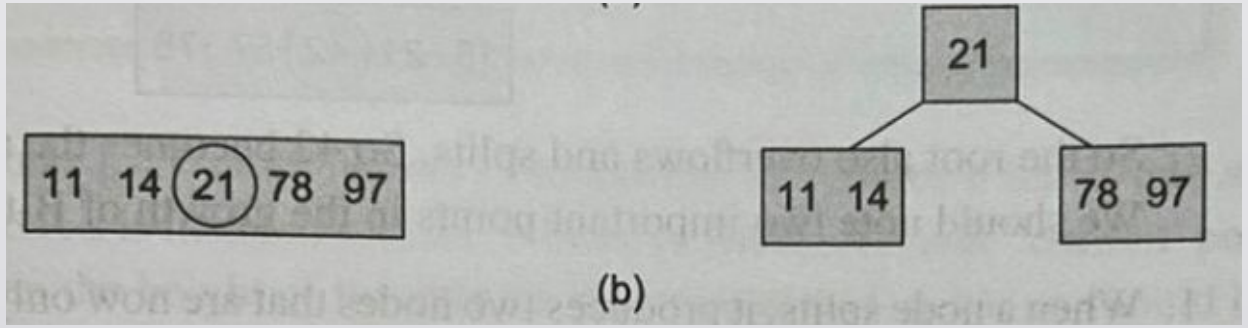
Example

- Build a B-tree of order 5 for the following data: 78, 21, 14, 11, 97, 85, 74, 63, 45, 42, 57, 20, 16, 19, 52, 30, 21.

a) First the numbers 78, 21, 14, and 11 are inserted.



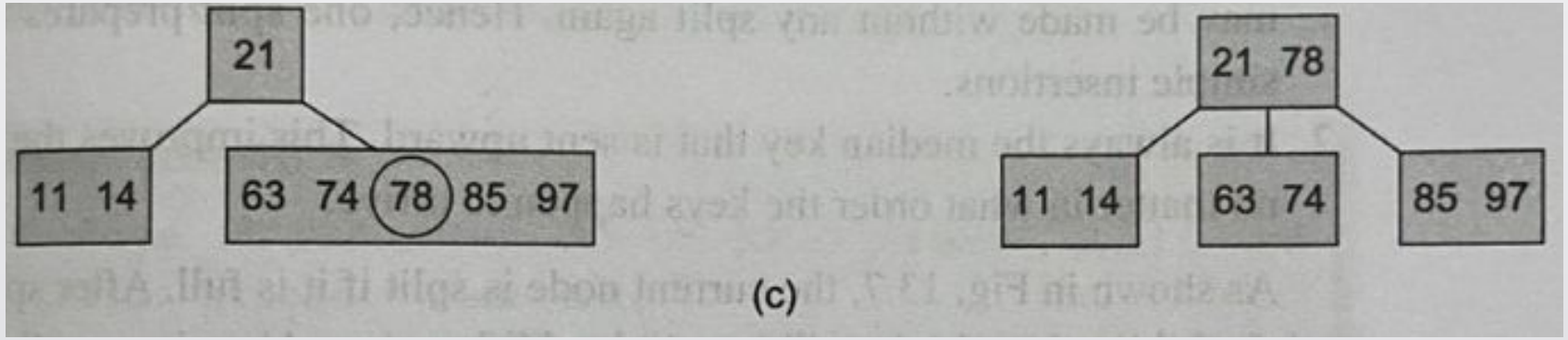
b) Then, 97 is inserted, an overflow occurs at 21, and the tree is split as



Example

- Build a B-tree of order 5 for the following data: 78, 21, 14, 11, 97, 85, 74, 63, 45, 42, 57, 20, 16, 19, 52, 30, 21.

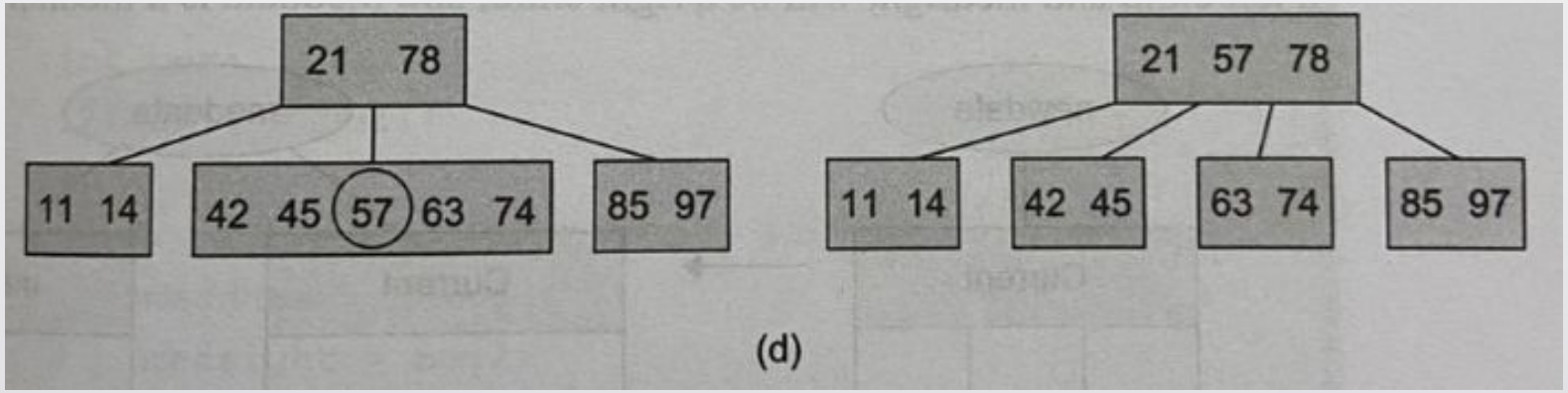
c) The numbers 85, 74, 63 are inserted and again the tree is split as shown



Example

- Build a B-tree of order 5 for the following data: 78, 21, 14, 11, 97, 85, 74, 63, 45, 42, 57, 20, 16, 19, 52, 30, 21.

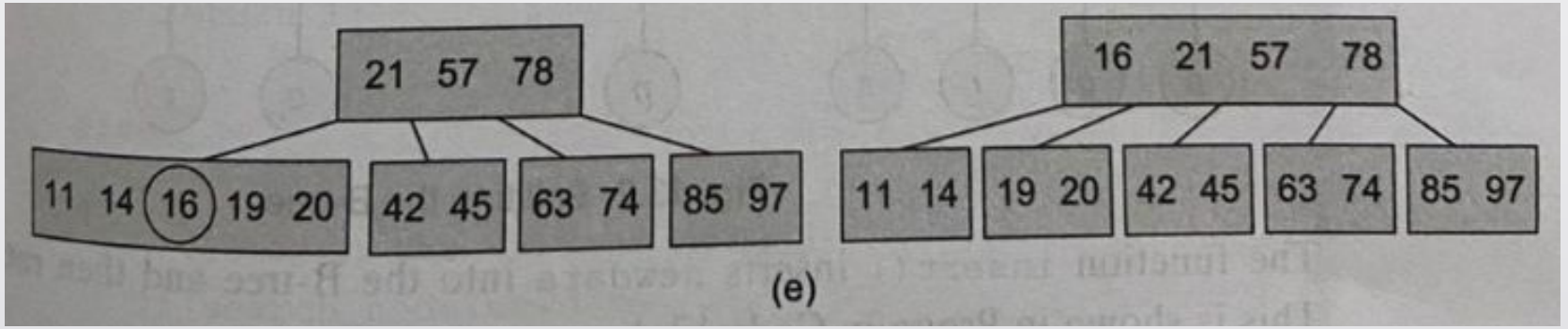
d) Below fig shows the split tree after insertion of 45, 42 and 57



Example

- Build a B-tree of order 5 for the following data: 78, 21, 14, 11, 97, 85, 74, 63, 45, 42, 57, 20, 16, 19, 52, 30, 21.

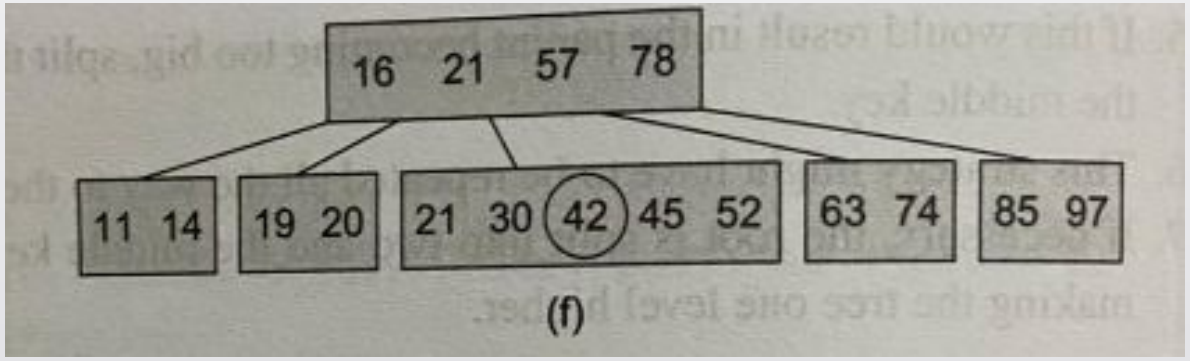
e) Below fig shows the split tree after insertion of 20,16, and 19



Example

- Build a B-tree of order 5 for the following data: 78, 21, 14, 11, 97, 85, 74, 63, 45, 42, 57, 20, 16, 19, 52, 30, 21.

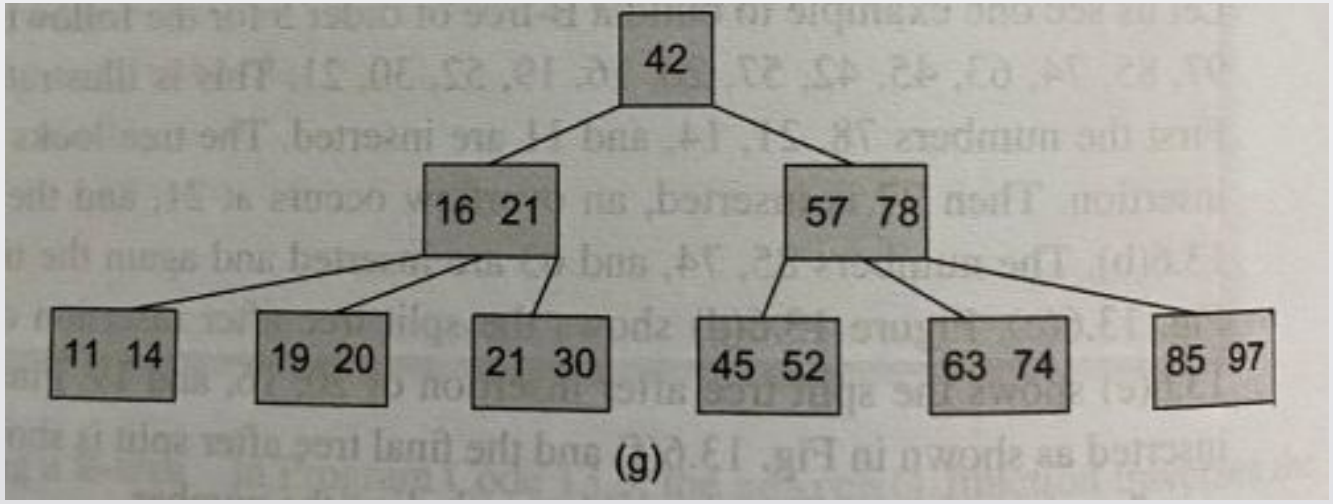
f) Finally 52, 30, 21 are inserted



Example

- Build a B-tree of order 5 for the following data: 78, 21, 14, 11, 97, 85, 74, 63, 45, 42, 57, 20, 16, 19, 52, 30, 21.

g) The final tree after split is



(g)

Deleting from a B-tree

- During insertion, the key always goes *into* a *leaf*. For deletion, if we wish to remove *from* a leaf, there are three possible ways mentioned as follows:

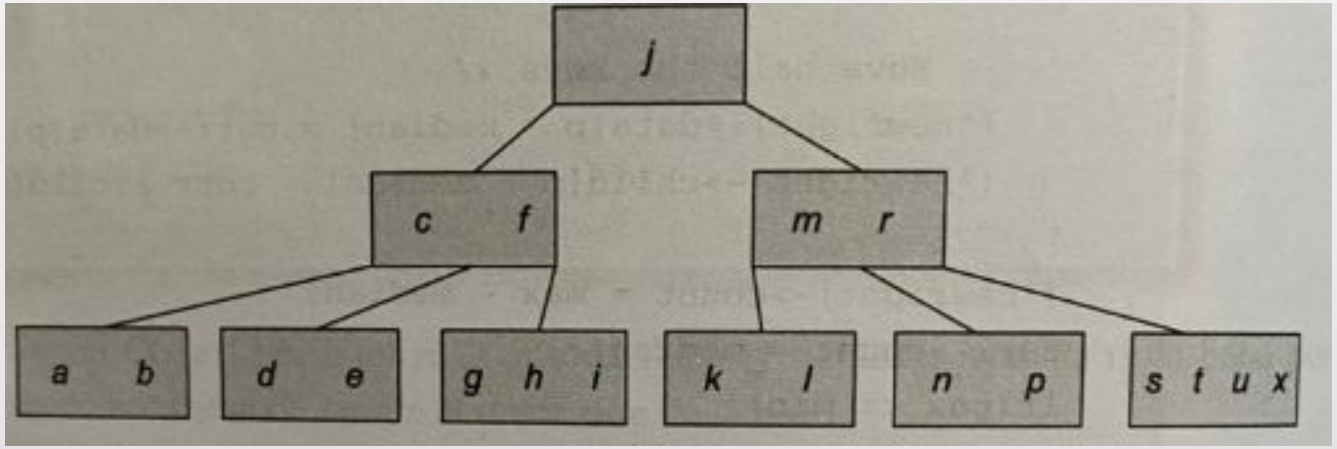
1. If the key is already in a leaf node, and removing it doesn't cause that leaf node to have too few keys, then simply remove the key to be deleted.
2. If the key is *not* in a leaf, then it is guaranteed (by the nature of a B-tree) that its predecessor or successor will be in a leaf -- in this case,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

Deleting from a B-tree (contd..)

3. If (1) or (2) conditions 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:
   a) 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.
   b) 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.

Example

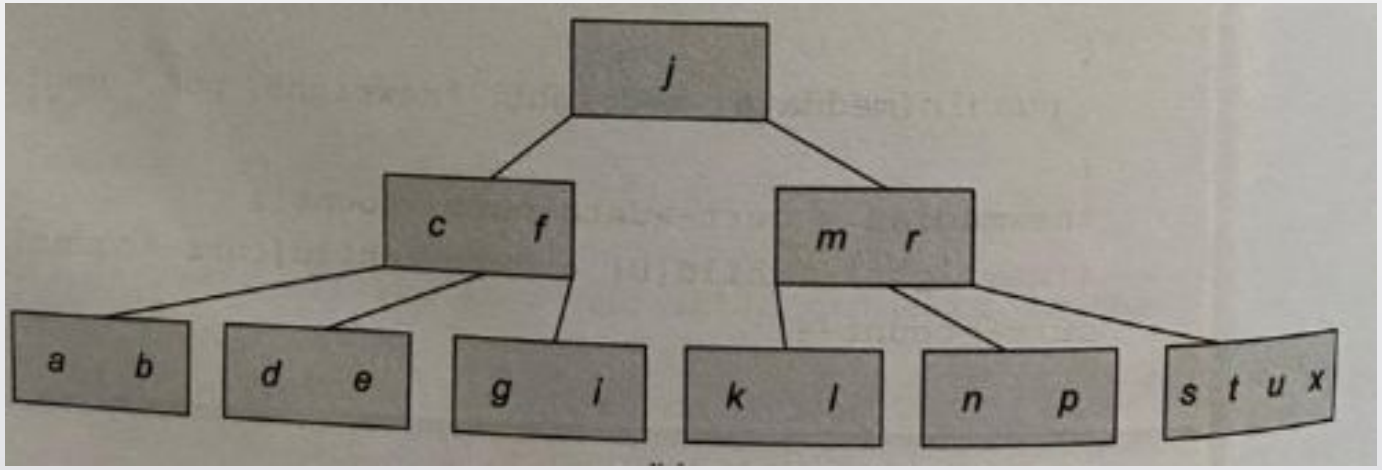
- Consider the following example.



- Now, Delete h.

Example

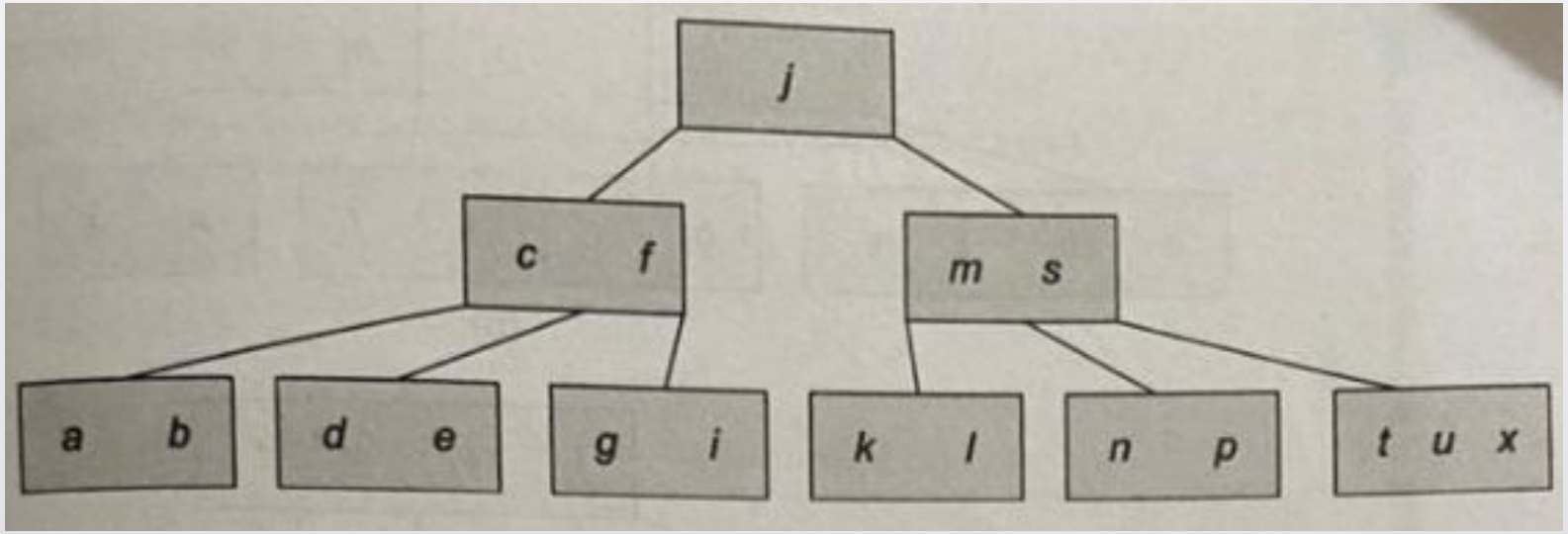
- After deleting h



- Now, delete r

Example

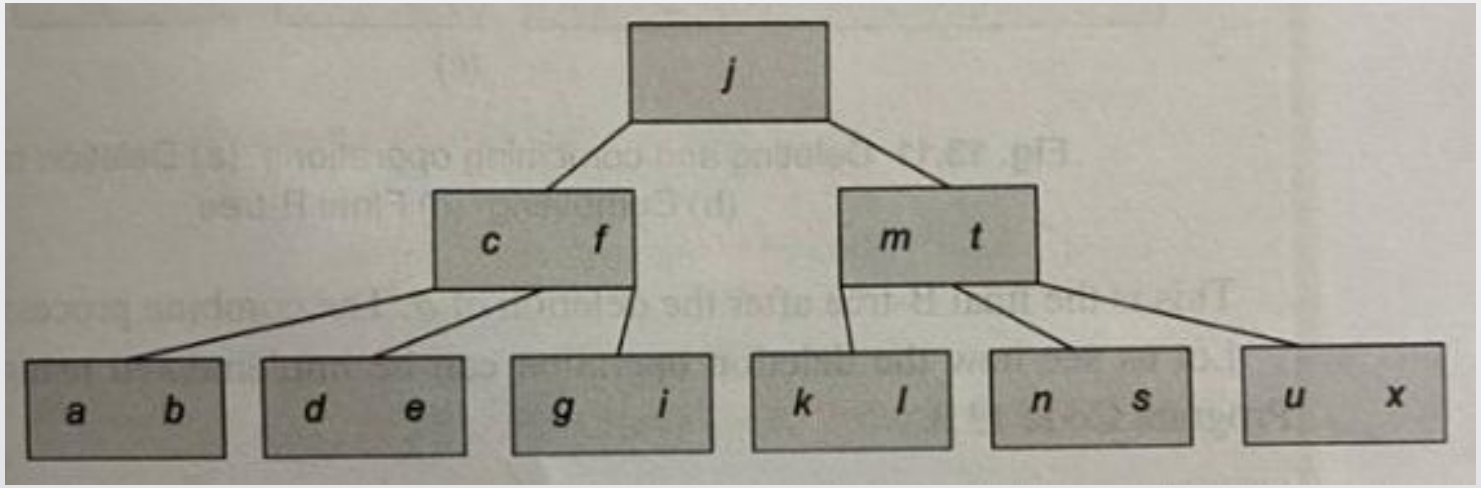
- Tree after deleting r and s is promoted.



- Now, delete p

Example

- Tree after p is deleted, s is moved down and t is moved up to the parent.

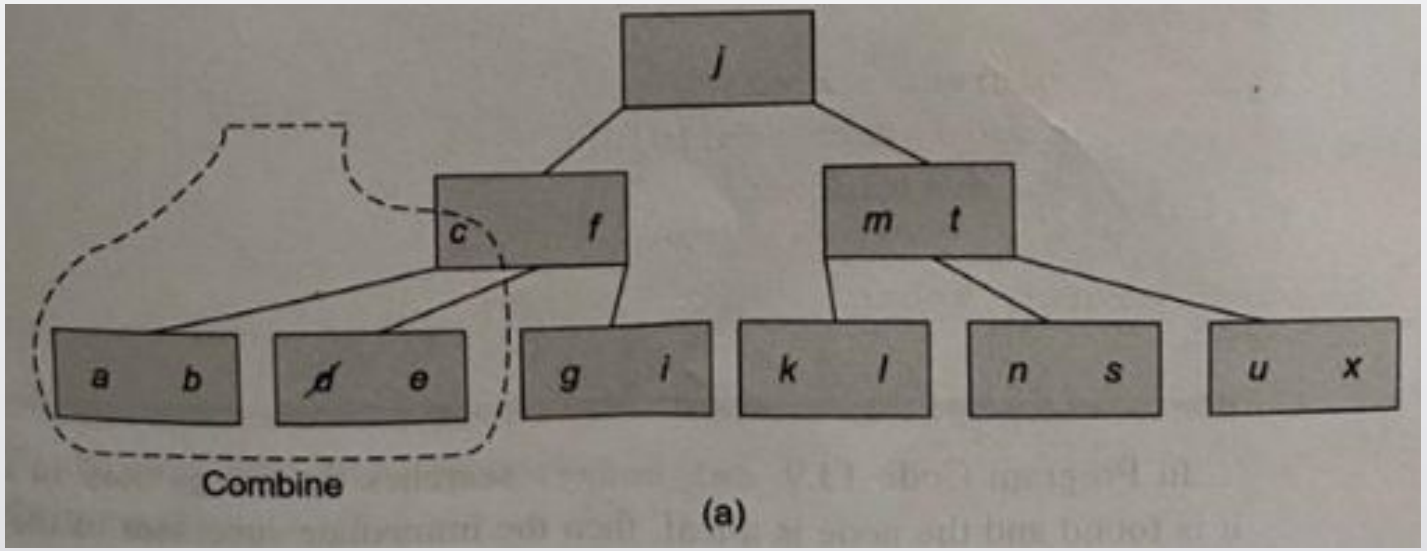


- Delete d

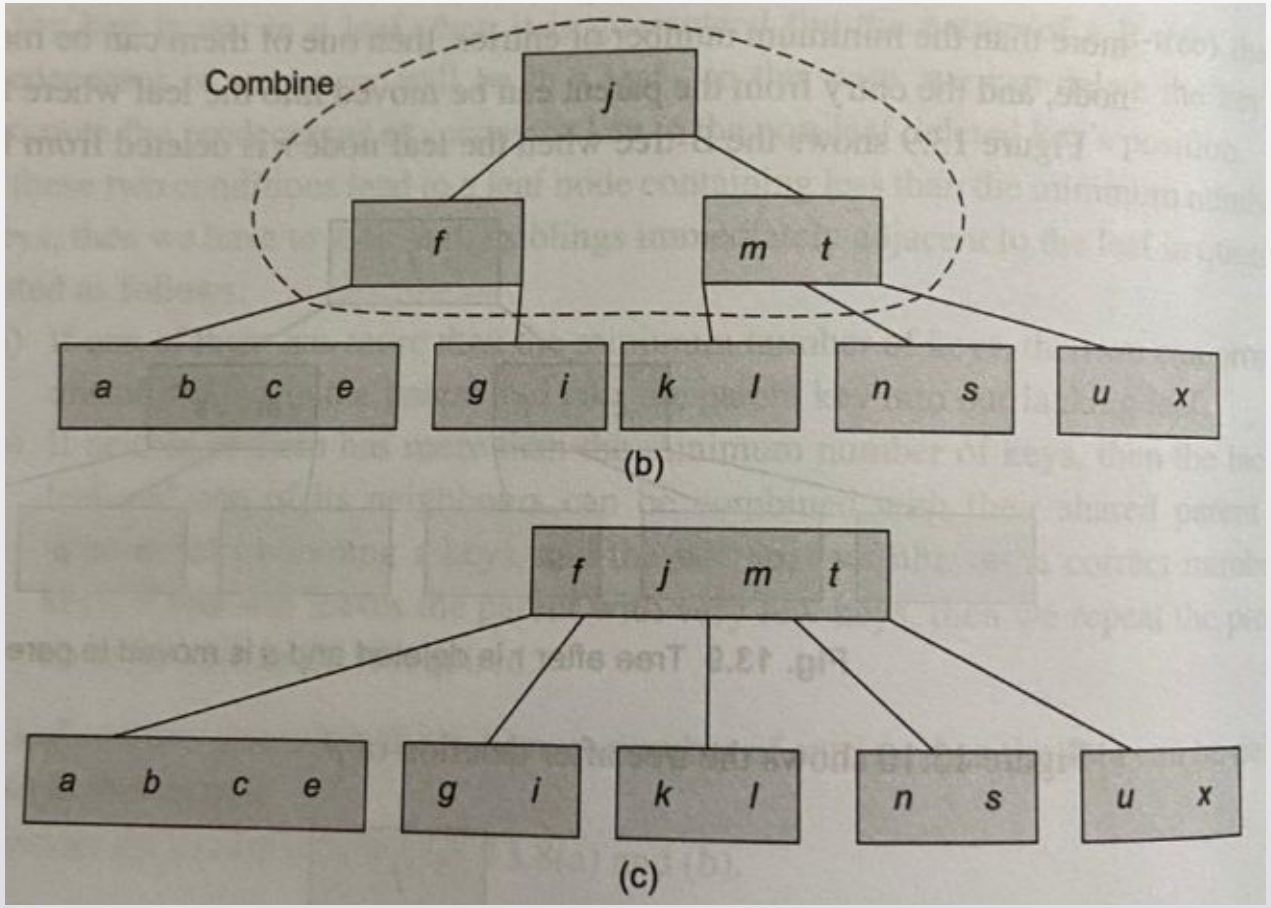
Example

- Deleting and combining operations

a) Deletion of node d b) combining c) Final B-Tree



Example



Complexity for Insertion

- Inserting a key into a B-tree of height *h* is done in a single pass down the tree and a single pass up the tree

Complexity:

where n is total no. of elements in B-Tree $O(h) = O(\log_t n)$

t is order

Deletion Complexity

- Basically downward pass:
  - Most of the keys are in the leaves – one downward pass
  - When deleting a key in internal node – may have to go one step up to replace the key with its predecessor or successor

Complexity $O(h) = O(\log_t n)$

where n is total no. of elements in B-Tree

t is order

B-Tree Analysis

- The maximum no. of items in a B-Tree or order m and height h can be:

Level	No. of keys
Root	m-1
Level 1	m(m-1)
Level 2	$m^2(m-1)$
:	:
Level h	$m^h(m-1)$

- So, total no. of items is

  $(1+m+m^2+m^3+\ldots\ldots+m^h)(m-1)= [(m^{h+1}-1)/(m-1)](m-1)=m^{h+1}-1$

- When m=5 and h=2, Max no. of items $=5^3-1=124$.

Reasons for using B-Trees

- When searching tables held on disk, the cost of each disk transfer is high but doesn't depend much on the amount of data transferred, especially if consecutive items are transferred
  - If we use a B-tree of order 101, say, we can transfer each node in one disk read operation
  - A B-tree of order 101 and height 3 can hold $101^4 - 1$ items (approximately 100 million) and any item can be accessed with 3 disc reads (assuming we hold the root in memory)
- If we take $m = 3$, we get a **2-3 tree**, in which non-leaf nodes have two or three children (i.e., one or two keys)
  - B-Trees are always balanced (since the leaves are all at the same level), so 2-3 trees make a good type of balanced tree

References

- *Varsha H. Patil. 2012. Data Structures using C++. Oxford University Press, Inc., USA*
- Gilberg, Richard F., and Behrouz A. Forouzan. *Data Structures: A pseudocode approach with C*. Nelson Education, 2004.