Database Management System
Prof. Partha Pratim Das
Department of Computer Science & Engineering
Indian Institute of Technology, Kharagpur

Lecture - 28
Indexing and Hashing/2: Indexing/2

Welcome to module 27 of Database Management Systems. We are discussing indexing and hashing mechanisms in a database and this is the second in that series.

(Refer Slide Time: 00:28)

Module Recap

PPD

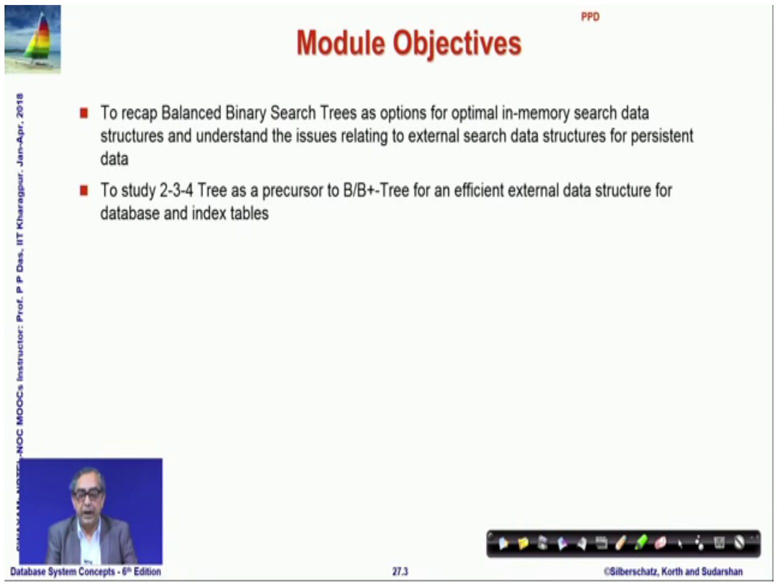
- Basic Concepts of Indexing
- Ordered Indices

SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P P Das, IIT Kharagpur. Jan-Apr, 2018

Database System Concepts - 6th Edition 27.2 ©Silberschatz, Korth and Sudarshan

In the last module, we have discussed the basic requirement of indexing and we have learnt about ordered indexes using primary index.

(Refer Slide Time: 00:42)



Module Objectives

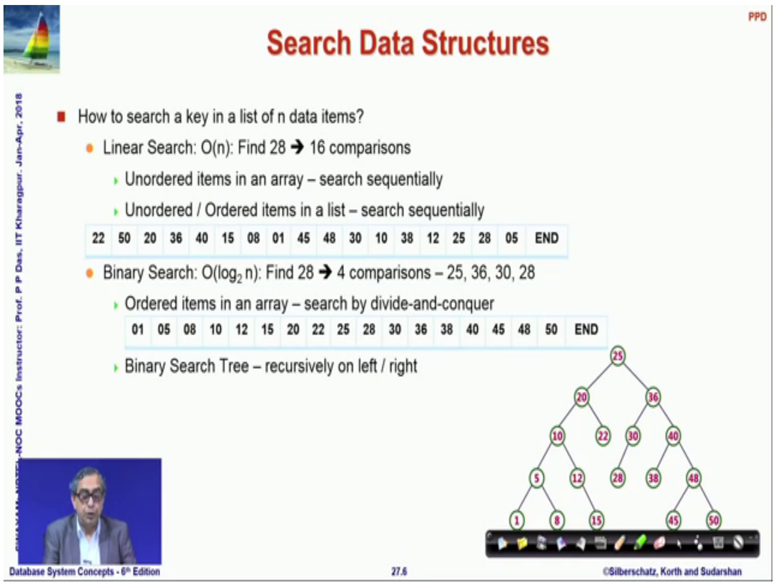
- To recap Balanced Binary Search Trees as options for optimal in-memory search data structures and understand the issues relating to external search data structures for persistent data
- To study 2-3-4 Tree as a precursor to B/B+-Tree for an efficient external data structure for database and index tables

PPD
SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P P Das, IIT Kharagpur. Jan-Apr, 2018
Database System Concepts - 6th Edition
27.3
©Silberschatz, Korth and Sudarshan

Which can be dense or parts and the multi level indexes. Now, in this module we would try to look for the basis of how indexing the index file structure can be very efficiently represent it. So, we will do a quick recap of our notions in algorithms goes.

Where we have talked about earlier balanced binary search trees I mean not in this particular course delivery, but I expect that you have gone through algorithms course where you have learnt about balanced binary search trees as options for optimal in memory search data structure and from that will try to understand the issues relating to external search data structures for persistent data and very specifically we will study two three four tree as a precursor to BB plus tree which is an very efficient external data structure for databases and index tables. So, these are the two topics to cover.

(Refer Slide Time: 01:47)



PPD

Search Data Structures

- How to search a key in a list of n data items?
  - Linear Search: O(n): Find 28 ➔ 16 comparisons
    - Unordered items in an array – search sequentially
    - Unordered / Ordered items in a list – search sequentially

22	50	20	36	40	15	08	01	45	48	30	10	38	12	25	28	05	END

  - Binary Search: O($\log_2$ n): Find 28 ➔ 4 comparisons – 25, 36, 30, 28
    - Ordered items in an array – search by divide-and-conquer

01	05	08	10	12	15	20	22	25	28	30	36	38	40	45	48	50	END

    - Binary Search Tree – recursively on left / right

NOC MOOCs Instructor: Prof. P P Das, IIT Kharagpur. Jan-Apr, 2018

Database System Concepts - 6th Edition 27.6 ©Silberschatz, Korth and Sudarshan

So, first let me quickly take you around with search data structure now I should warn you that here I am looking at we are looking at a little different kind of a problem here we are looking at search as it is performed in the algorithms course which mean that when you talk about data structures in algorithms course.

You typically talk of data structures which have two basic properties one they are volatile data structures transient that is they are created when the program starts and you operate on the data structure find queries and then as soon as your program ends they disappeared. So, they are not persistent data in contrast when you are dealing with database we are dealing with persistent data which stays even when no queries being performed.

And the second which is the consequence of the first is the data structure that you are study in algorithms work in memory. So, they work in a small limited space and they could be volatile whereas, the database the data structure required for databases has to reside in the disk. So, we have seen the tradeoff between the; on the storage hierarchy between memory and disk and other layers.

So, they will be brought to memory and then certain operations done and then possibly written back and so, on. So, there is similarity in terms of the strategies, but there is a very significant difference in that that because of the persistency the data structures that are used in the database application in the persistent data application has to work with a very different kind of cost. So, when we talk about cost of an algorithm say a search algorithm we will say that a search algorithm or a sort algorithm has a certain complexity VSSR you know the merge sort has the complexity order n login n and what it actually means.

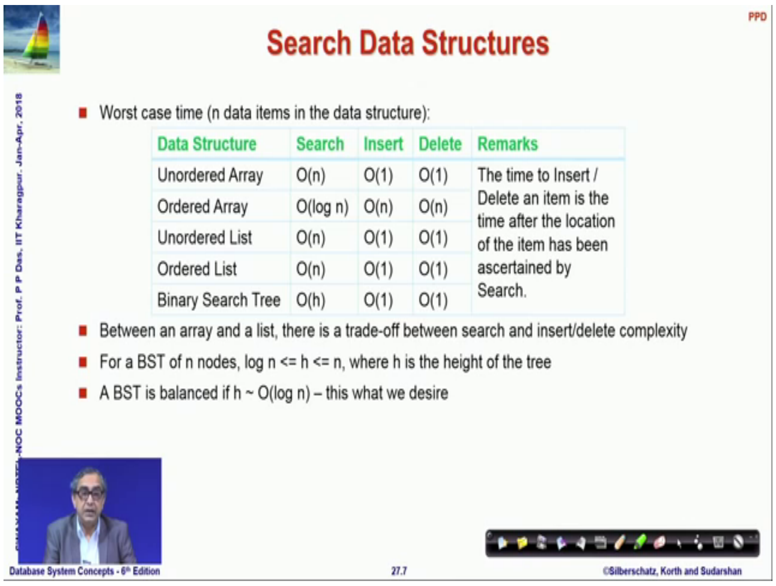
Is a number of comparisons you can estimate to do it in sorting n numbers is approximately n times log n, but when we talk about external data structure or disk base data structure then your cost may often not be the operations in the CPU like comparison or addition or assignment your cost will shift to actually the number of disk access is the page access is that you have to do, because as you have already noted that the cost of a disk access is much larger couple of orders larger compared to the basic cost of different processor operations.

So, I will start with this in this module I will start talking about data structures which we are found to be efficient in memory and then we will migrate to seeing how they can be used in a with simple extension in the as external data structures as well. So, what we have if you have given a to search a key in a list of n data items what are the different choices I could make use of a linear search either items could be in an array ordered or unordered in an array or they could be on a list either ordered or unordered I can search that list sequentially and find the data item.

So, I have just shown an example here there is a couple of data items given in that area and trying to find 28, there are 16 comparisons that I need to do and we all know that we can do much better if we keep the this item sorted in terms of in an increasing order or say decreasing order here it is increasing order and then we can do a binary search divide and conquer and I can find the same value 28.

Now, in just four comparisons and from that we have come to the also know that this whole binary search order can be easily represented in terms of a binary tree structure called the binary search tree.

(Refer Slide Time: 05:44)



Search Data Structures

- Worst case time (n data items in the data structure):

Data Structure	Search	Insert	Delete	Remarks
Unordered Array	O(n)	O(1)	O(1)	The time to Insert / Delete an item is the time after the location of the item has been ascertained by Search.
Ordered Array	O(log n)	O(n)	O(n)	
Unordered List	O(n)	O(1)	O(1)	
Ordered List	O(n)	O(1)	O(1)	
Binary Search Tree	O(h)	O(1)	O(1)	

- Between an array and a list, there is a trade-off between search and insert/delete complexity
- For a BST of n nodes, log n <= h <= n, where h is the height of the tree
- A BST is balanced if h ~ O(log n) – this what we desire

PPD

SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P P Das, IIT Kharagpur. Jan-Apr, 2018

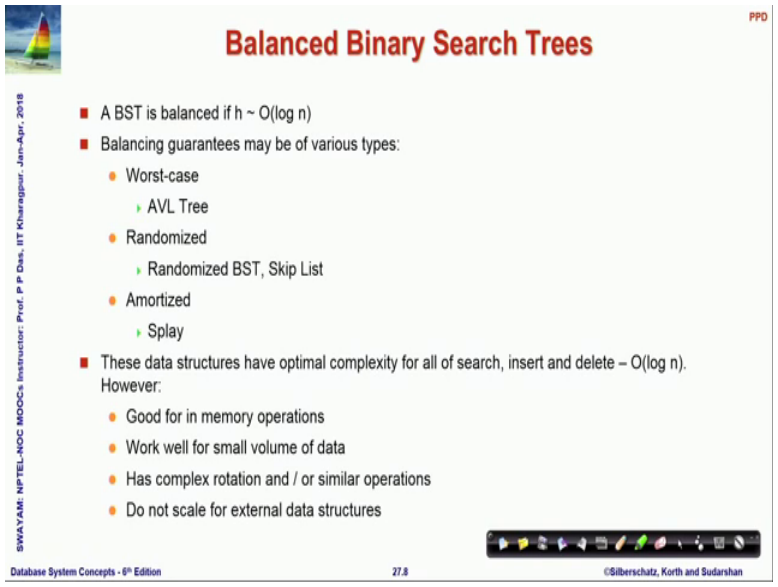
Database System Concepts - 6th Edition 27.7 ©Silberschatz, Korth and Sudarshan

So, if we compare worst case time here again here please keep in mind that we are talking about in memory data structure. So, here the time is primarily that of comparison. So, if you look at the different data structure like unordered array ordered array unordered list and ordered list and binary search tree and if you check the complexity of the three basic operations which we will need in that in a database application the research insert and delete you can find that the search. Usually, is order n unless you have a un ordered array I mean between array and list the search is order n unless you have an ordered array and you can do a binary search.

The insert the time that I show for insert or for delete is usually order one, because when I say insert is order one what it means its that after I have been able to search an item which I need to insert after I have been able to find its position, what is the additional time that you need to actually insert that item? So, that insert cost is often for unorder array and any kind of list is order one because you can just manipulate a couple of pointers and insert that, but if you are using an ordered array to make your search efficient then to insert you need a order n insertion because at the right place you need to move the elements to the right to make the space for the new element, because you need to maintain the ordering that the elements have.

So, kind of we find that between the array and the list there is kind of a trade off in terms of if you want to make search better you have ordered array and that degrades the insert delete complexity and vice versa. So, to take the benefit of both we the binary search tree is device to a you expect that the search will take the worst case order h cost which h is the height of the tree, because that is the maximum number of comparisons that we will need to reach that leaf node and a BST binary search tree if it is balanced then h would be of the order of log n and this is what we desire.

(Refer Slide Time: 07:58)



Balanced Binary Search Trees

- A BST is balanced if h ~ O(log n)
- Balancing guarantees may be of various types:
  - Worst-case
    - AVL Tree
  - Randomized
    - Randomized BST, Skip List
  - Amortized
    - Splay
- These data structures have optimal complexity for all of search, insert and delete – O(log n). However:
  - Good for in memory operations
  - Work well for small volume of data
  - Has complex rotation and / or similar operations
  - Do not scale for external data structures

PPD

SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P P Das, IIT Kharagpur. Jan-Apr, 2018

Database System Concepts - 6th Edition 27.8 ©Silberschatz, Korth and Sudarshan

So, if you look at BST is balanced h is of the order of log n I am not going into the theory of proving why balancing means h is of the order of log n or how this order of log n comes if you have doubts please refer to your algorithms book you will find plenty of that now that naturally now if a in the data structure if I am regularly inserting and deleting data.

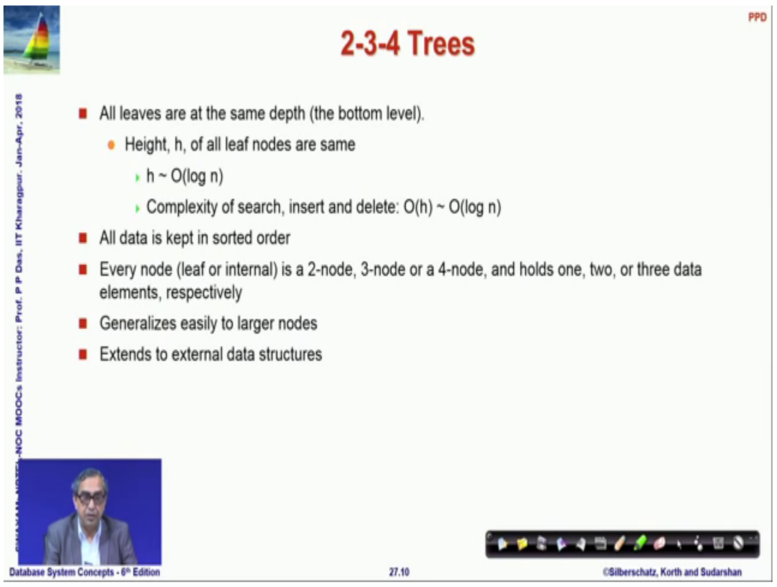
Then it is not guaranteed that it will remain balanced it might for example, if I am inserting the data in a in a say in increasing order in a in a binary search tree then every time the insertion will happen on the rightmost node and if. So, that will mean that I will have along the rightmost node I will have a long chain and therefore, it become like a linear list and therefore, any search in that will not remain optimally it will take order in time.

So, there as different strategies that you have studied in terms of balancing just to remind you there are strategies which give you the best possible worst case time of log n which is the famous AVL tree there are randomized strategies in terms of randomized BST skip list there are amortized strategies which say that I do not really care about what happens with a particular insertion search or deletion, but what I care is if I have done a large number of insert delete search operations on the array then on the average what it should be balance on the average it should be of ordered log n.

So, we have seen all of these different strategies and. So, in an order in an attempt to generalize them for external data structure we note that these are typically good for in memory operations these are good worked well; when you deal with a small volume of data I mean you may be that may still be large, but is small in the sense that the whole data fits into the memory and you can manipulate muscles the whole data in memory and it of course, all these many many of these algorithms.

Particularly, the AVL tree and quite a bit of the randomized BST have complex rotation operations that need to be performed the order different random generators need to be involved in the randomized BST or skip list. So, there are other complexities in this whole factor comparison cost is not the only cost that you have and in the simple thing is they do not skill, what external data structures they are not optimized in terms of minimizing or optimizing the disc accesses or they do not scale to millions and millions of entries and so, on.

(Refer Slide Time: 10:38)



2-3-4 Trees

- All leaves are at the same depth (the bottom level).
  - Height, h, of all leaf nodes are same
    - h ~ O(log n)
    - Complexity of search, insert and delete: O(h) ~ O(log n)
- All data is kept in sorted order
- Every node (leaf or internal) is a 2-node, 3-node or a 4-node, and holds one, two, or three data elements, respectively
- Generalizes easily to larger nodes
- Extends to external data structures

PPD

SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P P Das, IIT Kharagpur. Jan-Apr, 2018

Database System Concepts - 6th Edition 27.10 ©Silberschatz, Korth and Sudarshan

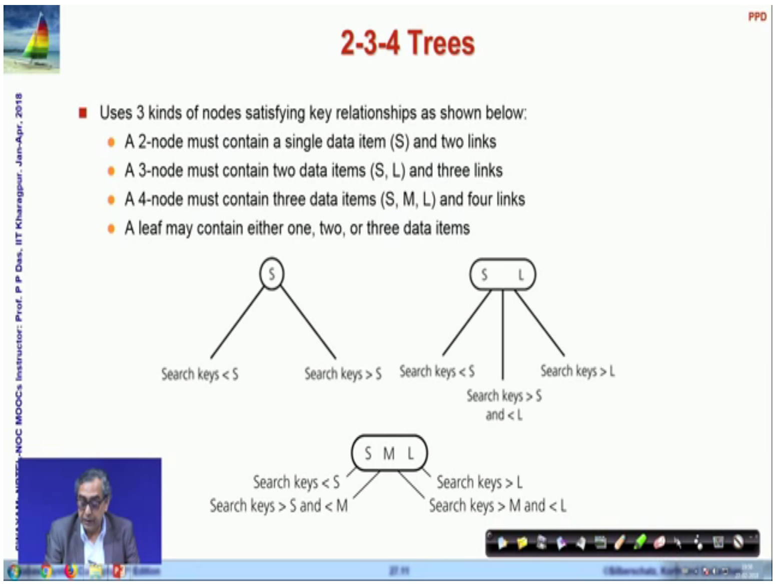
So, you need to look at a different approach and this different approach in the day in terms of database application database structures is called B plus tree and what I am going to discuss is an in memory version of that which is a 2, 3, 4 trees. So, that we can understand the basic principle of such search data structure which can work with external data with this data, but first understand them in as an in memory version.

So, what is a 2, 3, 4 tree a 2, 3, 4 tree in contrast to other BSTs or in contrast to the typical BSTs where you know every operation needs the height of the tree to be balanced because some leaves could happen at a much deeper level some could be at a much shallow level in a 2, 3, 4 `tree all leaves always are at the same level the same depth the bottom level.

So, height h is the height of all the leaf nodes and that is guaranteed to be of order of log n, if the tree has n number of nodes the complexity of search, delete and insert all are order h that is a consequence of a constant height h or or a fixed height h that given n that is maintained all data are kept in a sorted order. So, if you do a in order traversal of the tree you will get the data in the sorted order, but the (Refer Time: 12:13) difference is in contrast to the BST where every node is a binary node is a is a (Refer Time: 12:19) one key and has two children here every node either a leaf node or an internal node every node is one of the three types it can either be a 2-node or a 3-node or a 4-node.

A 2-node holds one, data 3-node once holds 2 data and 4-node holds 3 data and there they give the name get the name 2-node 3-node and 4-node based on the number of children that they can have the 2-node can have 2 children, 3-node 3 children and 4-node 4 children. They can generalized easily to larger nodes which can have a large number of different types of nodes and it extends very naturally to external data structure. So, that is with the basic about the 2, 3, 4 tree.

(Refer Slide Time: 13:05)


2-3-4 Trees

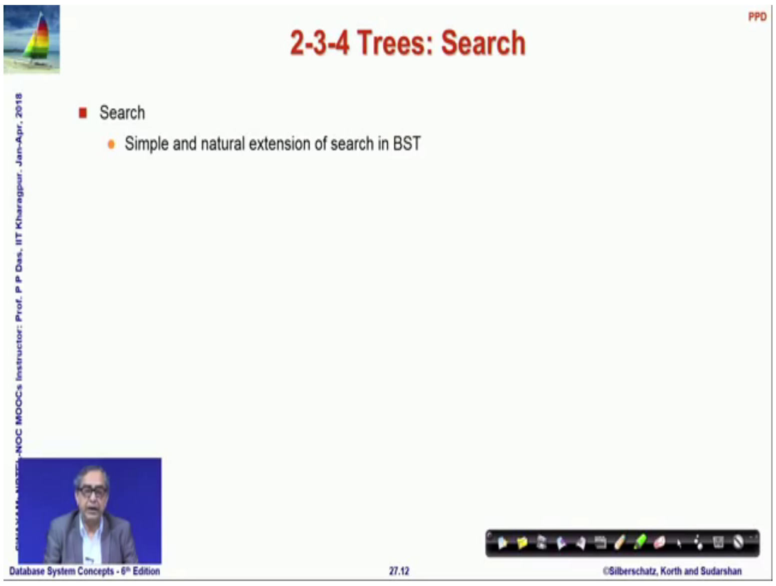
- Uses 3 kinds of nodes satisfying key relationships as shown below:
  - A 2-node must contain a single data item (S) and two links
  - A 3-node must contain two data items (S, L) and three links
  - A 4-node must contain three data items (S, M, L) and four links
  - A leaf may contain either one, two, or three data items

So, let us just go through it in little bit more detail it uses three kinds of node as I have said and now let me just show you. So, if you are talking about a 2-node. So, this is a 2-node. So, there is one data item S and all search keys which are less than S around this side all search keys which are on greater than S around this side.

We are just for the simplicity of discussion where I am assuming that all keys in this data structure are unique. So, there is no repeated keys the repeated keys can be handled in a very easy manner. So, this is one type of node a second type of node is a 3-node which must contain two data items S we are calling it S and L and three links the first is less than S the last is greater than L and the middle is greater than s, but less than L which means actually what we enforce in terms of the two keys that exist at the node S must be less than L.

So, values less then L go on one link values between S and L go on the middle link and values greater than n go on the third link. Similarly, if I have a 4-node here I have three values where S is less than M is less than l. So, now, you can understand why the acronyms S, M and L small, medium and large. So, on and we have four links the first link gives you values less than S second is between S and M third between M and L and forth greater than L. So, these are the or three different types of nodes that a 2, 3, 4 tree can support and if it is a leaf node then it can be it can contain either 1, 2 or 3 get items. So, let us go forward.

(Refer Slide Time: 15:01)



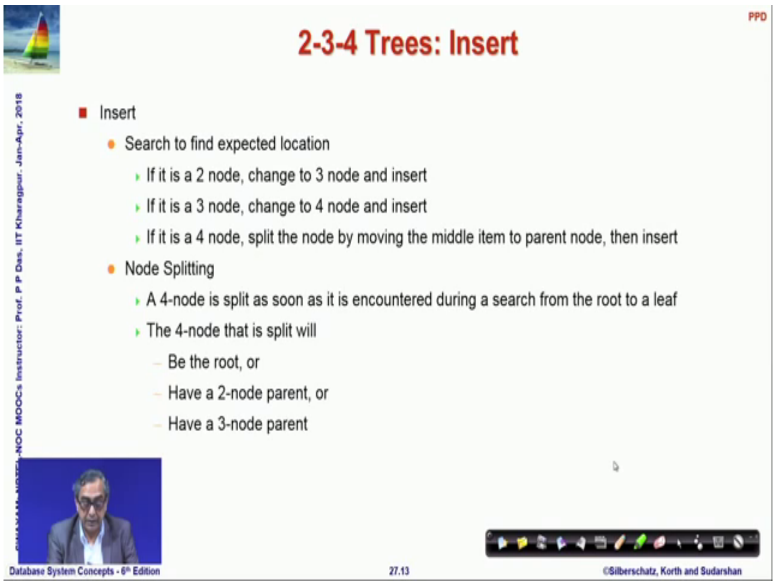
2-3-4 Trees: Search

- Search
  - Simple and natural extension of search in BST

PPD
SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P P Das, IIT Kharagpur. Jan-Apr, 2018
Database System Concepts - 6th Edition
27.12
©Silberschatz, Korth and Sudarshan

Now, to search to search is a simple extension of BST you know how to search in a BST you start with the route see whether the given key to search is greater than the key at the route if it is greater you go to right if it is less you go to left and you do the same thing here for a 2-node for a 3-node all that you will need to find out is between S and L whether it is less than L then less than S then you take the leftmost whether it is greater than L then you take the rightmost if it is between S and L you take the middle similar strategy you do for 4-node and search is a simple extension of the BST algorithm. So, you can suddenly work it out.

(Refer Slide Time: 15:38)



2-3-4 Trees: Insert

- Insert
  - Search to find expected location
    - If it is a 2 node, change to 3 node and insert
    - If it is a 3 node, change to 4 node and insert
    - If it is a 4 node, split the node by moving the middle item to parent node, then insert
  - Node Splitting
    - A 4-node is split as soon as it is encountered during a search from the root to a leaf
    - The 4-node that is split will
      - Be the root, or
      - Have a 2-node parent, or
      - Have a 3-node parent

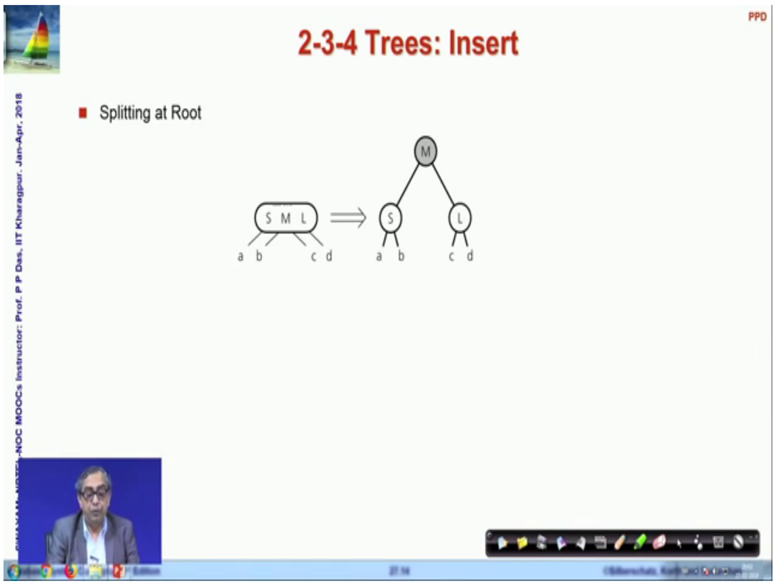
PPD
SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P P Das, IIT Kharagpur. Jan-Apr, 2018
Database System Concepts - 6th Edition
27.13
©Silberschatz, Korth and Sudarshan

Now, what we will need to do in terms of an insert, insert is very interesting. So, for insert first you search and find the expected location where you are expecting it. So, that is not there. So, you will expect. So, now, what are the possibility? Possibilities where you have found the expected location that node could be a 2-node if it is a 2-node all that you simply need to do is change that where 3-node inserts the second item if it is a 2-node it has only one item. So, just insert this new item there and making it into a 3-node good.

In the second case if you find the location if you find that it is a 3-node. So, it has two items you just change it to a 4-node and insert this is as a third item; obviously, when you insert you will have to decide has to whether where you should insert that depends on whether you are given key is greater than the key that already existed or smaller than that and so, on. Now, the question is what happens; when if you locate the place to be insert itself is already a 4-node. Now, naturally you do not have anything bigger than a 4-node. So, if it is a 4-node then all that you need is to split that node you have to split the node. So, you have a 4-node.

So, you have a 4-node. So, you have you have a data one here you have data 2 here, you have data 3 here and you are you know that the your d has to get inserted in this. So, you cannot insert it by changing the node type because this is a maximum allowed. So, all that you want to do is to change that into at different structure and that structure is called a node splitting structure. So, we will we will see in the next slide as to how this is done and we will see how different splitting sequence can happen and what will happen when a 4-node will split when it was a root or the different kinds of parents that it has let us just go there.

(Refer Slide Time: 17:42)

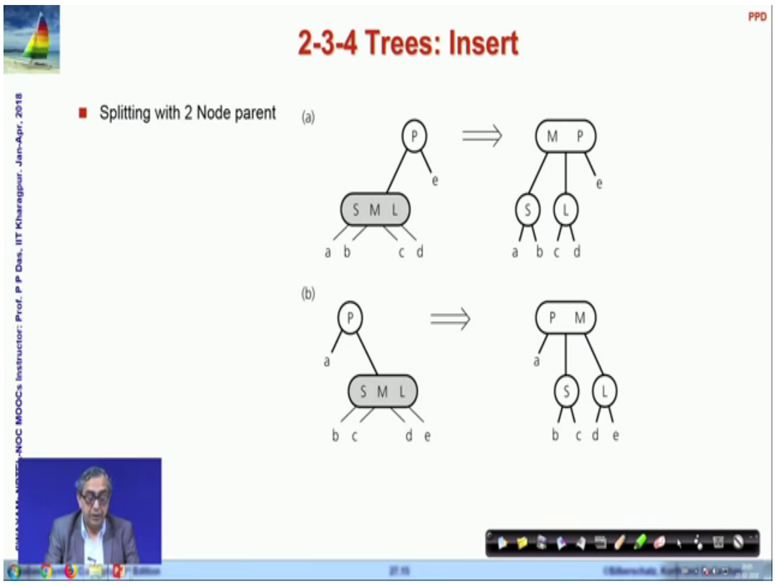

2-3-4 Trees: Insert

- Splitting at Root

So, what we are saying is the suppose you have a 4-node which is a root now splitting that is actually doing this. So, you have to convince yourself that enough 2, 3, 4 tree what we have already assumed; whether I represent this or I represent this are algorithmically their equivalent. So, a single root 4-node and a such a structure of your 3, 2-nodes are equivalent why is it? So, for example, if you are looking say if you are looking for a here then it is it say it is less than s. So, you come here now if you are looking for the same a here what happens a is less than S. So, it is it must be less than M remember S is less than M is less than L.

So, a is less than M. So, you take this part it is less than S to you come here let us take a case of c lets say c. So, if it is c you should come here, now if you check here c if it is falling on this link; that means, it is greater than M and it is less than L. So, is greater than M you come here it is less than L. So, you come here. So, you reach the same link. So, you can see that actually a 2, 3, 4 tree is not a unique representation depending on the requirement I can replace 4-nodes in terms of other 2-nodes and actually create a new tree configuration. So, if it is at the root I can get rid of a 4-node and replace it by this equivalent tree.

(Refer Slide Time: 19:27)


2-3-4 Trees: Insert

- Splitting with 2 Node parent

Now, suppose there are the 4-node is not at the root it is somewhere else. So, what are the possibilities the; if it is not at the root it must have a parent now the parent could be a 2-node. So, if it is a 2-node then these are the two possibilities if it is a 2-node, then. So, this is a parent node which is a 2-node. So, then the 4-node could be a left child of that or it could be a right child of that if it is a left child, then we use split take the middle item and insert it in the parent and by that process a parent becomes a 3-node from a 2-node and make these become two different 2-nodes.

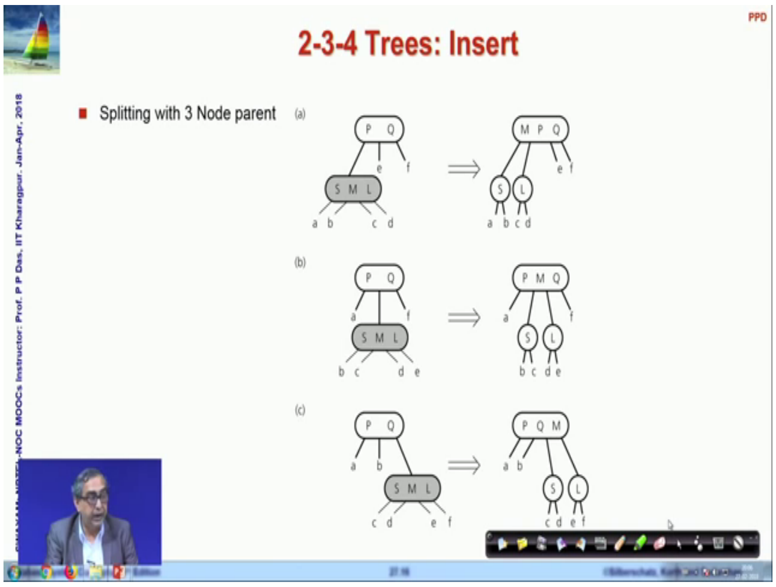
Again the way I was just explaining in the in the previous slide you can convince yourself that this structures are equivalent for example, if I am looking for d let us say if I am looking for d in this tree.

So, if I am looking for d that is greater than l. So, how do I arrive here now since this is a left child? So, it must be placed than P, otherwise it could not have occurred on this side. So, it is less than P. So, d is less than P d is greater than l. So, given that if I search for d here so, it is less than P and since it is we I also have from this the d is greater than M, otherwise it would not have come to the; this third link this forth link it would have been elsewhere. So, I know that d is greater than M.

So, and it is less than P. So, if I combined this 2, then d has to go on this middle ink, because it is greater than M and less than P and then I have it is greater than L. So, it has to be on the right. So, it comes to the right position.

So, in this way you can convince yourself in every search case that if the parent is a 2-node, then you can equivalently split the 4-node and make an insertion in the parent 2-node converted into 3-node and get rid of the 4-node altogether that lives us with.

(Refer Slide Time: 21:45)



2-3-4 Trees: Insert

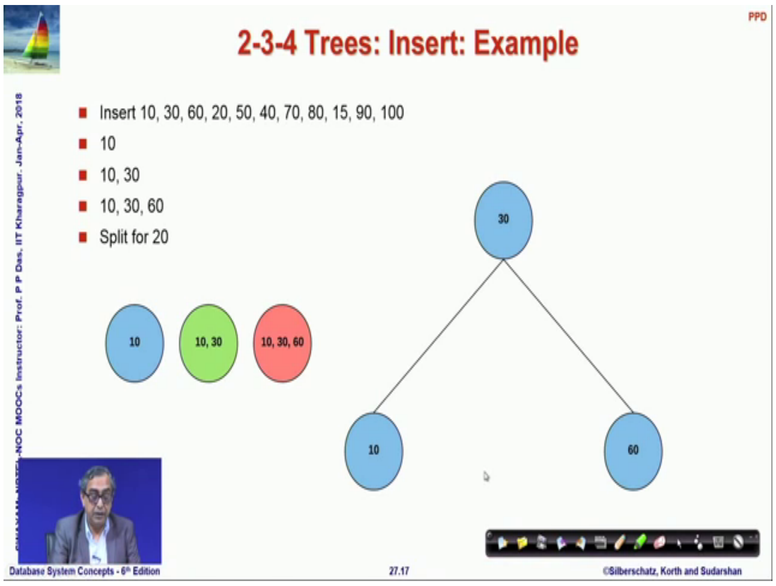
- Splitting with 3 Node parent

One other case which is if the parent is a 3-node naturally if it is the parent is a 3-node then there are three possibilities. Now, because your 4-node could be a left child a middle child or a right child and in every case you do the same thing you split the 4-node take the middle item put it to the parents. So, parent converts from 3-node to 4-node. Now and your rest of the split and pointed adjustments had done. So, that you get rid of this.

So, what happens in this process in and like the earlier ones here you do not get rid of the 4-nodes altogether, because in replacing one 4-node your creating another 4-node, but in the process what is happening the 4-node is moving up it is it is going one level up.

So, again what you will do is in recursively the new parent 4-node parent will again be split and I just split if it is it is parent that is parent of the parent if that parent is also a 3-node, then that will become a 4-node this will continue till your root becomes a 4-node and we know that when root becomes a 4-node I can always change it to a configuration of 3, 2-nodes. So, in this process as we have shown that we can actually get rid of the 4-nodes in the whole of the 2, 3, 4 tree when it is required.

(Refer Slide Time: 23:03)



2-3-4 Trees: Insert: Example

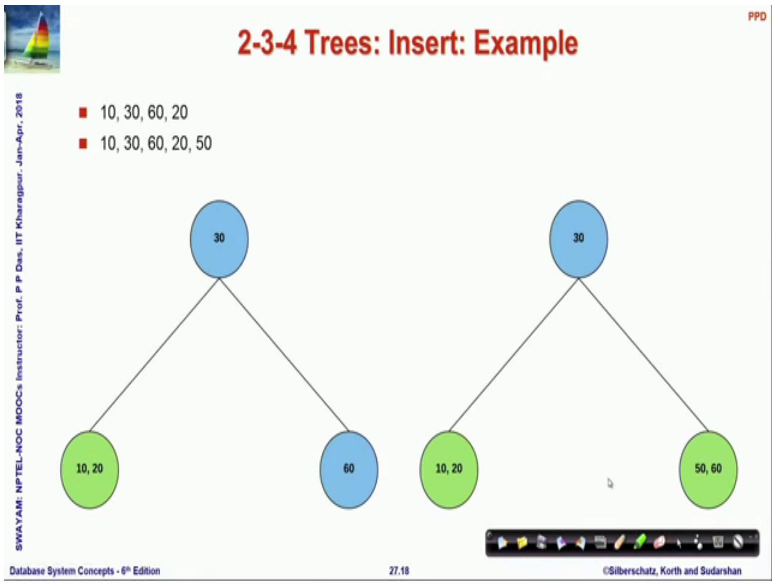
- Insert 10, 30, 60, 20, 50, 40, 70, 80, 15, 90, 100
- 10
- 10, 30
- 10, 30, 60
- Split for 20

10
10, 30
10, 30, 60
30
10
60
PPD
NPTEL-NOC MOOCs Instructor: Prof. P P Das, IIT Kharagpur. Jan-Apr, 2018
Database System Concepts - 6th Edition
27.17
©Silberschatz, Korth and Sudarshan

So, the basic strategy is very simple that will keep on constructing the 2, 3, 4 tree and whenever we come across a 4-node for the first time we will split that and rearrange. So, that I can get rid of it so, here what I show you is a basically an insert sequence over the next couple of slides where which is trying to insert the data in this following order starting with an empty 2, 3, 4 tree. So, we first insert 10. So, you get this then we insert 30 that is here. So, 2-node becomes. So, the here the convention that I am I am following is blue is a 2-node green is a 3-node and red is a 4-node. So, then you insert sixty it becomes a 4-node and as soon as it becomes a 4-node the next element to be entered is 20.

But, before that this 4-node will have to be split and this is the case of splitting at the root, because this is this has no child yet. So, you split and you get the middle moves to the top here as 30, then you have two children 10 and 60 and after the splitting you move on to actually inserting the intended 20 into it.

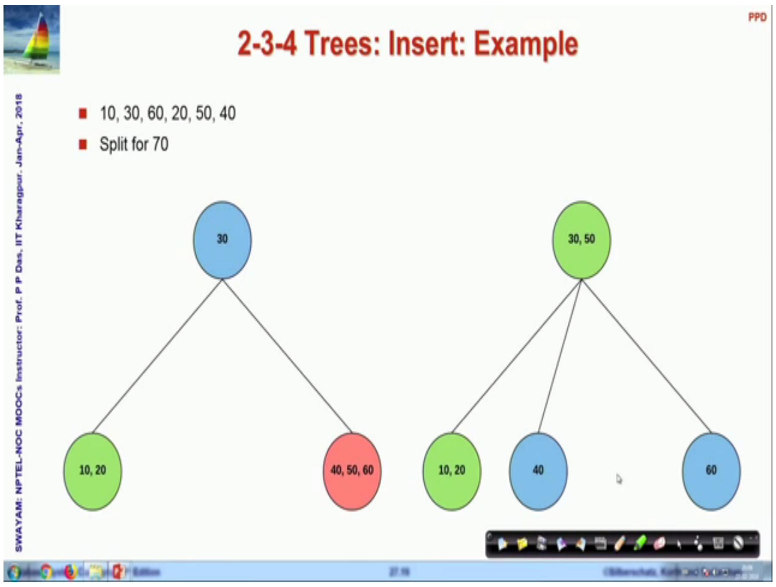
(Refer Slide Time: 24:19)


2-3-4 Trees: Insert: Example

- 10, 30, 60, 20
- 10, 30, 60, 20, 50

So, 20 gets inserted on this side 20 is inserted on this side, because a smaller than this and comes here. So, this 2-node becomes a 3-node then you insert 50 which goes on this side which goes on this side and gets inserted here.

(Refer Slide Time: 24:41)

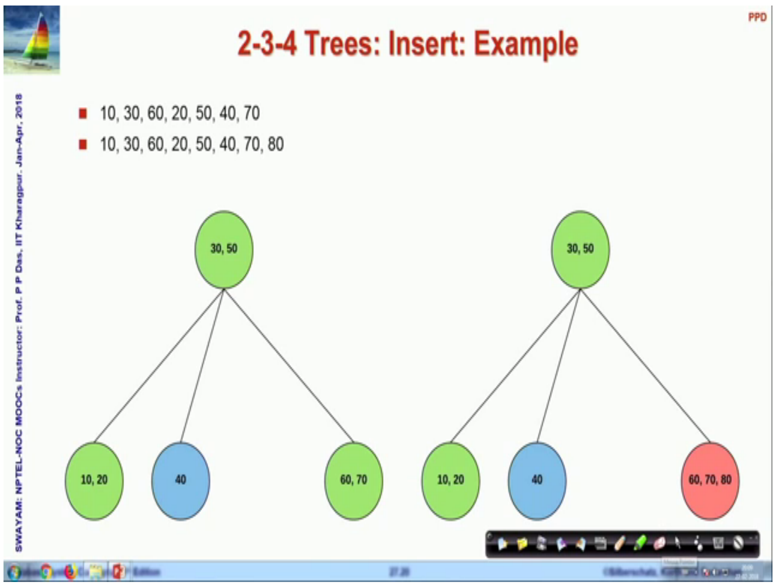

2-3-4 Trees: Insert: Example

- 10, 30, 60, 20, 50, 40
- Split for 70

So, your insertion continues your next to insert is 40 which gets the inserted here it becomes a 4-node and as soon as it becomes a 4-node and before the next insertion of 70 can happen you need to do a split. So, you do a split.

If you do a split then your 50 moves to the parent this becomes a 3-node. Now and your 40 and 60 becomes two 2-node children as this. So, it is a same information is represented, but now in this new 2, 3, 4 tree you do not have any 4-node. So, in you can go ahead and insert 70.

(Refer Slide Time: 25:19)



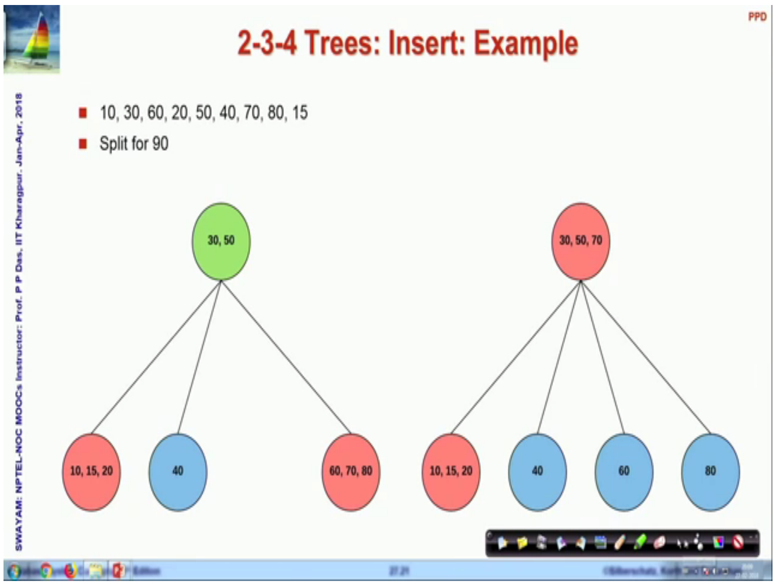
2-3-4 Trees: Insert: Example

- 10, 30, 60, 20, 50, 40, 70
- 10, 30, 60, 20, 50, 40, 70, 80

PPD
SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P P Das, IIT Kharagpur. Jan-Apr, 2018
30, 50
10, 20
40
60, 70
30, 50
10, 20
40
60, 70, 80

So, insert 70, 70 gets inserted here the 2-node becomes 3-node the, this is done. So, then you insert 80, 80 gets inserted here is greater than it comes here and this becomes a 4-node and. So, naturally the next would be two insert 15. So, 15 goes in on to the left. So, 15 has come in here which becomes a 4-node.

(Refer Slide Time: 25:43)



2-3-4 Trees: Insert: Example

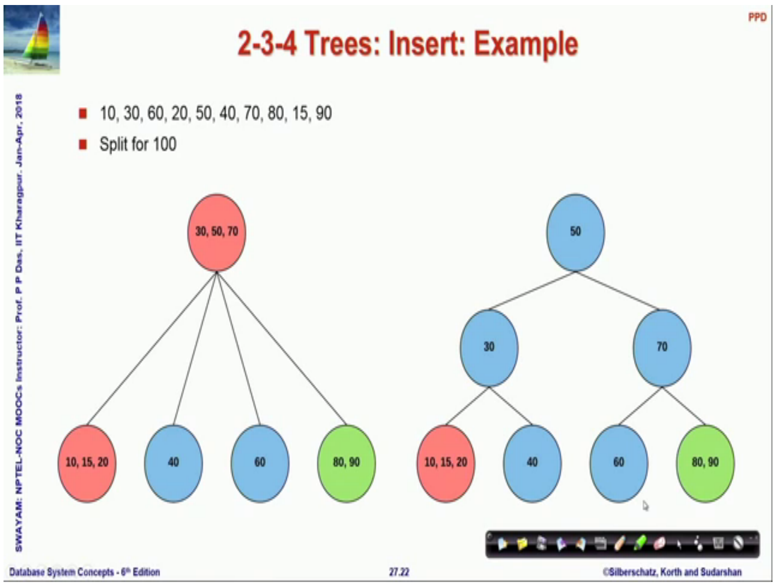
- 10, 30, 60, 20, 50, 40, 70, 80, 15
- Split for 90

PPD
SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P P Das, IIT Kharagpur. Jan-Apr, 2018
30, 50
10, 15, 20
40
60, 70, 80
30, 50, 70
10, 15, 20
40
60
80
27.21

And the next is to insert 90. So, which has to get inserted here it should have got inserted here. So, this is already a 4-node. So, you need to split.

So, in split and as you split you get 40 40 was already there you get 60 and 80 and the 70, that existed in the middle goes to the root and your root becomes a 4-node.

(Refer Slide Time: 26:27)



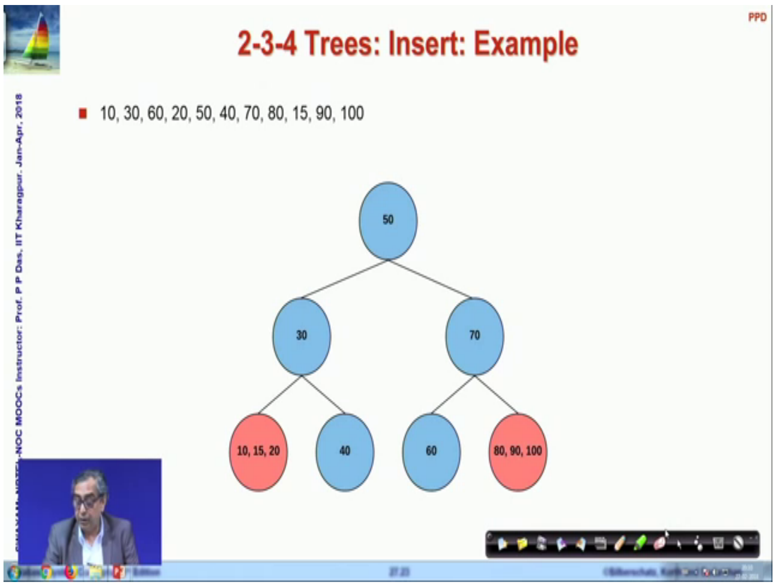
2-3-4 Trees: Insert: Example

- 10, 30, 60, 20, 50, 40, 70, 80, 15, 90
- Split for 100

PPD
SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P P Das, IIT Kharagpur. Jan-Apr, 2018
30, 50, 70
10, 15, 20
40
60
80, 90
50
30
70
10, 15, 20
40
60
80, 90
Database System Concepts - 6th Edition
27.22
©Silberschatz, Korth and Sudarshan

So, this is the configuration you move on to your 90 gets inserted. Now, you have to insert 100. So, you before that this your root has become a 4-node. So, you do a split. So, as you do the split this is a new configuration that has happened and this is what has resulted from the split of this 4-node at the root.

(Refer Slide Time: 26:59)



2-3-4 Trees: Insert: Example

- 10, 30, 60, 20, 50, 40, 70, 80, 15, 90, 100

And then naturally you can go ahead and insert 100 and 100 is now inserted. Here, you could do adjustments of changing this this 4-node into by splitting it and moving it onwards I have not shown that, but if you now go back if you just now go back on on these this whole process and you will see that specifically that we claim that all leaf nodes would be in the same level. So, you can see that initial three trees all are at the same level, then leaf and their level 0 and then when you have had this split then the leaf nodes 10 and 60 are both at level once.

So, we can see that when actually you split at the root you add one level to all the leaf nodes and that is the only time your height changes. So, beyond that you look at this the level has not changed I am going to the next the level has not changed instruments to be height 1 level does not change height 1 does not change does not change till the left here and then we have a case again of splitting a 4-node at the root.

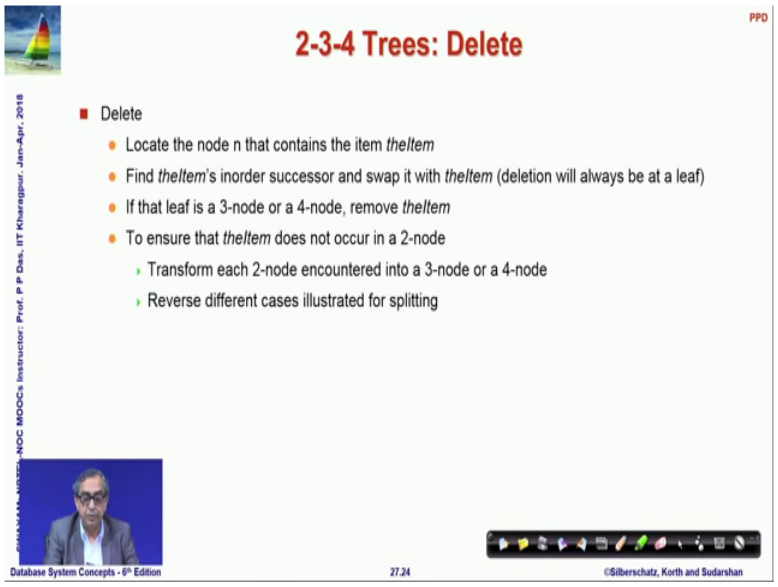
When the one level has been added to all the; so, all the leaf nodes where at level 1 now all of them are at level 2. So, in a 2, 3, 4 tree you achieve this invariance of all leaf nodes being at the same level by maintaining that the only time the height changes is when you split a 4-node at the root and add one level uniformly to all of them.

And then rest of the logic is quite simple that these are here you can do actually follow the same logic as of the binary search tree analysis that if there are n n items here; then the maximum height could actually be order log n; because we though all nodes are not binary, but the nodes that are not binary actually have more data.

So, they will be they will the height will always be log n or less than that of course, there is an issue of concluding the complexity, because now you will argue that there are 3-nodes or 4-nodes where more than one comparison is required to decide about the node, but the counter argument to that is even if that be the case even if you have along the path of the tree from the root to any leaf node.

Even if all of the nodes are of are 4-node that cannot happen as you have seen because you will keep on splitting and distributing that, but if all of them are or 4-node also then what will add is simply a factor of three two rather additional with the log n and the overall complexity remains to be log n will go.

(Refer Slide Time: 29:59)



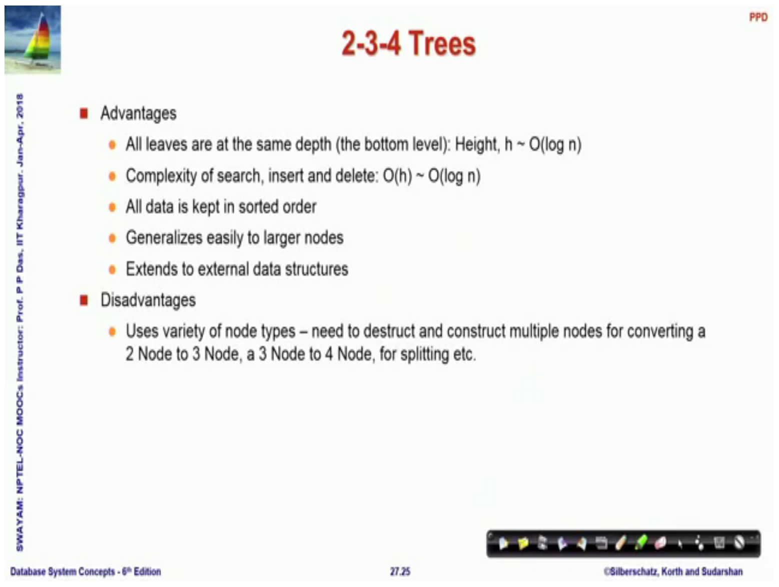
2-3-4 Trees: Delete

- Delete
  - Locate the node n that contains the item *theItem*
  - Find *theItem*'s inorder successor and swap it with *theItem* (deletion will always be at a leaf)
  - If that leaf is a 3-node or a 4-node, remove *theItem*
  - To ensure that *theItem* does not occur in a 2-node
    - Transform each 2-node encountered into a 3-node or a 4-node
    - Reverse different cases illustrated for splitting

Database System Concepts - 6th Edition 27.24 ©Silberschatz, Korth and Sudarshan

Now, if you have to delete you have to do very similar operations I have not shown the details, but you look at the node and then actually find the in order successor for that and swap it with the item and then you can do the other arrangements of collapsing the nodes as we have done the splitting of nodes. Now I have to do the collapsing of nodes following the same river structure and leave that as an exercise to you just work it out at home.

(Refer Slide Time: 30:28)



2-3-4 Trees

- Advantages
  - All leaves are at the same depth (the bottom level): Height, h ~ O(log n)
  - Complexity of search, insert and delete: O(h) ~ O(log n)
  - All data is kept in sorted order
  - Generalizes easily to larger nodes
  - Extends to external data structures
- Disadvantages
  - Uses variety of node types – need to destruct and construct multiple nodes for converting a 2 Node to 3 Node, a 3 Node to 4 Node, for splitting etc.

PPD

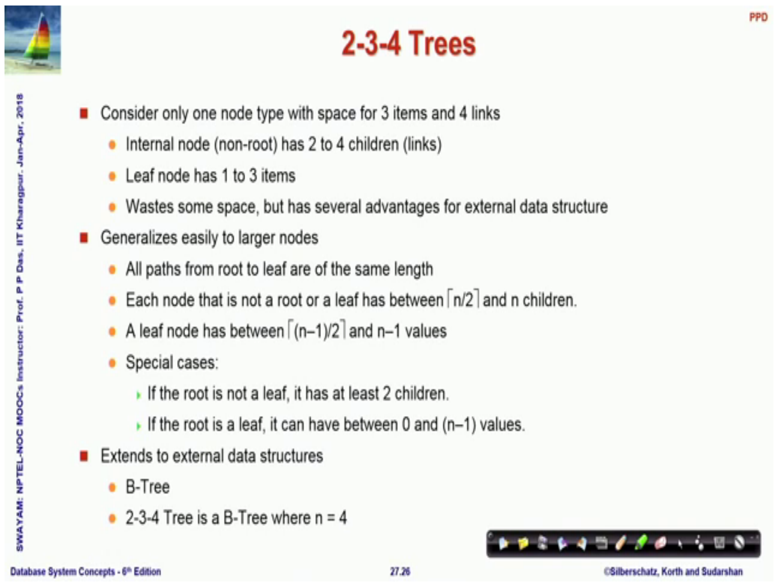
SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P P Das, IIT Kharagpur. Jan-Apr, 2018

Database System Concepts - 6th Edition 27.25 ©Silberschatz, Korth and Sudarshan

So, if you look at if the 2, 3, 4 tree and there is a time to justify why we are doing this all leaves are the same depth which is a great advantage the height is order log n always complexity of search insert delete all are order log n all data kept in sorted order it generalizes easily to larger nodes are and extends to external data structure. So, what you mean by larger nodes, let us let us look at that a little bit before that of course, 2, 3, 4 tree has a major disadvantage compared to binary research trees of the other kinds because it uses a variety of node types. So, you I mean when you change from a say 2-node to a 3-node you actually if you think in programming terms you have lot of additional cost, because you need to distract the 2-node and create a 3-node.

If you split a 4-node and create 3, 2-nodes as required, then you have to distract the 4-node and create 3, 2-nodes. So, there are lot of over rights in terms of that.

(Refer Slide Time: 31:32)



2-3-4 Trees

- Consider only one node type with space for 3 items and 4 links
  - Internal node (non-root) has 2 to 4 children (links)
  - Leaf node has 1 to 3 items
  - Wastes some space, but has several advantages for external data structure
- Generalizes easily to larger nodes
  - All paths from root to leaf are of the same length
  - Each node that is not a root or a leaf has between $\lceil n/2 \rceil$ and n children.
  - A leaf node has between $\lceil (n-1)/2 \rceil$ and n–1 values
  - Special cases:
    - If the root is not a leaf, it has at least 2 children.
    - If the root is a leaf, it can have between 0 and (n–1) values.
- Extends to external data structures
  - B-Tree
  - 2-3-4 Tree is a B-Tree where n = 4

PPD

SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P P Das, IIT Kharagpur. Jan-Apr, 2018

Database System Concepts - 6th Edition 27.26 ©Silberschatz, Korth and Sudarshan

So, what leaf if we if we just simplify that process and if you assume that, there is only one node type which has enough space for three items and four links we do not have two I mean we functionally there will be 2-node 3-node 4-node, but physically let them with a same type of node. So, any internal node can have what is specification? So, there the same type any internal node has 2 to 4 children and a leaf node has 1 to 3 items that is what all that we are saying.

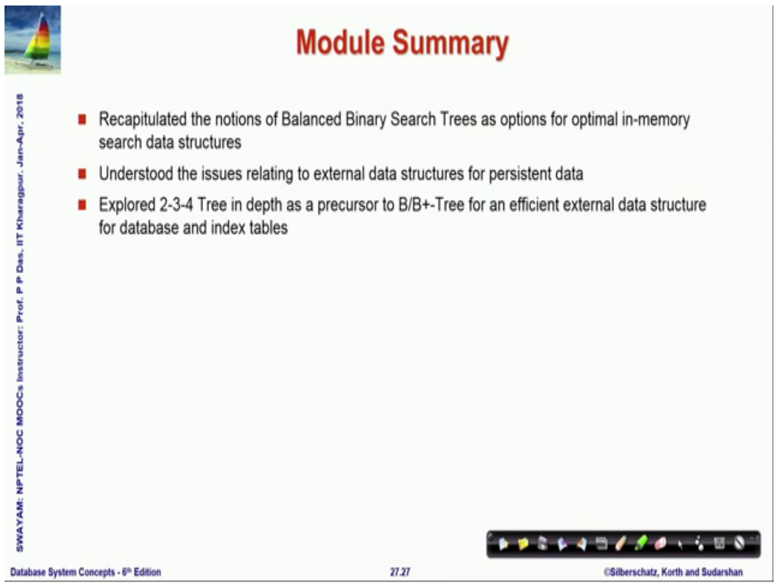
So, by doing this we will waste some space, but we have several advantages particularly when you look at this for external data structure. So, what will happen is if I can think about 2, 3, 4 tree in that manner then I can; obviously, generalize that it is not necessary that I will have to restrict myself at 4 links, 4 children, I can do more than that.

So, in general I can say that there are a node has n children and it is each node is not a leaf not a root or a leaf will have between n by 2 and n children. So, put n as 4 you will find that it becomes 2, 3, 4 tree and a leaf node will have n minus 1 by 2 which is 1 and for enough n being 4 and n minus 1 or 3 values which is what did you have.

So, if you just generalize from 2, 3, 4 to n by 2 to n and have a a container have a node container which can have either n by 2 or n by 2 plus 1 or n by 2 plus 2 or maximum up to n minus 1 data items and corresponding number of children then we will be very easily be able to arrange for a similar data structure which will have all the nodes all the leaf nodes at the same level and. So, this is the structure which is which extends very easily and this is a fundamental structure of what he says a B-tree or a b plus tree will see the differences shortly.

But this is a basic notion and the strategies of node splitting and node merging in case of deletion and the algorithm of insertion deletion that we have discussed here will simply get generalize in case of B-tree.

(Refer Slide Time: 33:57)



Module Summary

- Recapitulated the notions of Balanced Binary Search Trees as options for optimal in-memory search data structures
- Understood the issues relating to external data structures for persistent data
- Explored 2-3-4 Tree in depth as a precursor to B/B+-Tree for an efficient external data structure for database and index tables

SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P P Das, IIT Kharagpur. Jan-Apr, 2018

Database System Concepts - 6th Edition 27.27 ©Silberschatz, Korth and Sudarshan

When we go to the next module so, we have recapitulate in on the balanced binary search tree and we introduced a the notion of a 2, 3, 4 tree which is a precursor to b plus tree B-tree which is which are efficient external data structures and will be covered in the next module.