15CSE374
INTRODUCTION TO DATA STRUCTURES AND ALGORITHMS

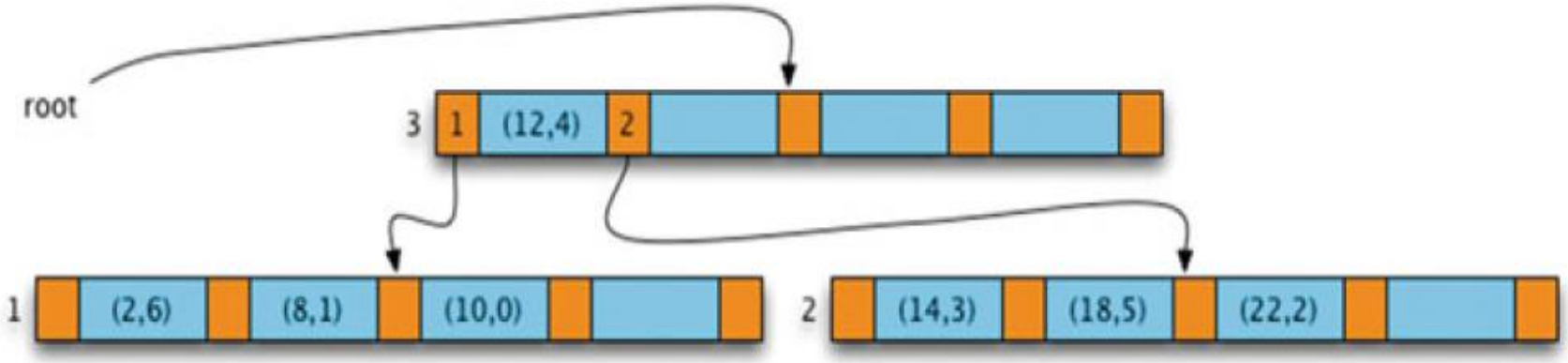
Sarath tv

Last Lecture

- Rotations
- B-Tree

B Tree Organization

- Each node in a B-Tree contains pointers to other nodes and items in an alternating sequence.
- The items in a node are arranged sequentially in order of their keys.
- A pointer to the left of an item points to another B-Tree node that contains items that are all less than the item to the right of the pointer.
- A pointer to the right of an item points to a node where all the items are greater than the item.



- There will always be one more pointer than items in a node.
- Degree of a B-Tree is the minimum number of items that a B-Tree node may contain, except for the root node.
- Capacity of the node is always twice its degree.
- For our example degree is 2 and capacity is 4.

The requirements of a B-Tree are

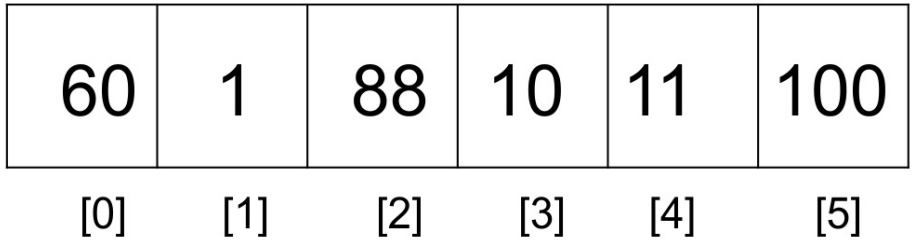
- Every node except the root node must contain between degree and 2*degree items.
- Every node contains one more pointer than the number of items in the node.
- The items within a B-Tree node are ordered in ascending (or descending) order.
- All nodes have their items in the same order, either ascending or descending.
- The items in the subtree to the left of an item are all less than that item.
- The items in the subtree to the right of an item are all greater than that item.

Searching

- Important operations for all data structures is searching for the elements from the stored data.
- Important operation for sorting.
- The sorting algorithm will be fast if the searching algorithm is efficient.
- Whether the items about to be searched have already been sorted or not,
- Searching algorithms are categorized into two broad types:
  - The searching algorithm is applied to the list of items that are already sorted; that is, applied to the ordered set of items
  - The searching algorithm is applied to the unordered set of items, which are not sorted

Linear search

- Find out a given item from the stored data.
- If the searched item is available in the stored list then it returns the **index position** where it is located, or **else** it returns that the item is **not found**.
- The simplest approach to search for an item in a list is the linear search method, in which we **look** for items **one by one** in the whole list.



- The preceding list has elements that are accessible through the list index. To find an element in the list, we employ the linear searching technique.
- This technique traverses the list of elements by using the index to move from the beginning of the list to the end.
- Each element is examined, and if it does not match the search item, the next item is examined.
- By hopping from one item to the next, the list is traversed sequentially.

Unordered linear search

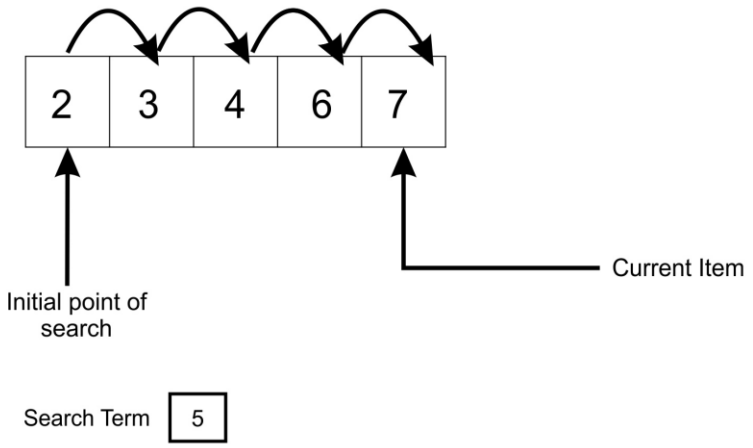
- Consider an example list that contains elements 60, 1, 88, 10, and 100—an unordered list. The items in the list have no order by magnitude.
- To perform a search operation on such a list, one proceeds from the very first item and compares that with the search item.
- If the search item is not matched then the next element in the list is examined. This continues till we reach the last element in the list or until a match is found.

```
def search(unordered_list, term):
    unordered_list_size = len(unordered_list)
    for i in range(unordered_list_size):
        if term == unordered_list[i]:
            return i

    return None
```

Ordered linear search

- Another case in a linear search is when the list elements have been sorted; then our search algorithm can be improved. Assuming the elements have been sorted in ascending order, the search operation can take advantage of the ordered nature of the list to make the search more efficient.




```
def search(ordered_list, term):
    ordered_list_size = len(ordered_list)
    for i in range(ordered_list_size):
        if term == ordered_list[i]:
            return i
        elif ordered_list[i] > term:
            return None

    return None
```

Binary search

- To find elements within a **sorted** array or list; thus, the binary search algorithm finds a given item from the given sorted list of items
- It starts searching the item by dividing the given list by half. If the search item is smaller than the middle value then it will look for the searched item only in the first half of the list, and if the search item is greater than the middle value it will only look at the second half of the list. We repeat the same process every time until we find the search item or we have checked the whole list.
- O(log n)

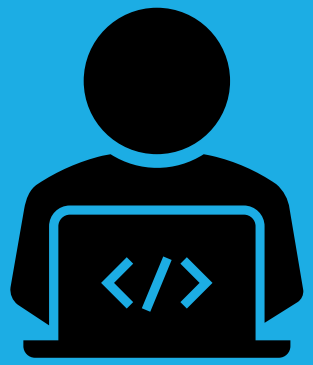
- Suppose we have a book of 1,000 pages, and we want to reach the page number 250. We know that every book has its pages numbered sequentially from 1 upwards. So, according to the binary search analogy, we first check the search item 250 which is less than the 500 (which is the midpoint of the book). Thus, we search the required page only in the first half of the book. We again see the midpoint of the first half of the book, that is, using page 500 as a reference we find the midpoint, that is, 250. That brings us closer to finding the 250th page. And then we find the required page in the book.

If we want to search 43 in the given list

since 43>37
So look only at second half

since 43<47
so now look only at first half of
this list

The search item 43 is found.
Function will returns the index position.



THANK YOU!!!!!

- B-Tree
- https://youtu.be/C_q5ccN84C8
- https://youtu.be/TOb1tuEZ2X4
- KD tree
- https://youtu.be/XG4zpiJAkD4
- https://www.geeksforgeeks.org/k-dimensional-tree/