CS/COE 1501
www.cs.pitt.edu/~xiaozhong/

B-Tree B+Trees
Huffman coding
A B-tree is a self-balancing tree data structure that keeps data sorted and allows searches, sequential access, insertions, and deletions in logarithmic time. [Wikipedia]
B-Tree Insertion

- If the node contains fewer than the maximum legal number of elements, then there is room for the new element. Insert the new element in the node, keeping the node's elements ordered.
- Otherwise the node is full, evenly split it into two nodes so:
  - A single median is chosen from among the leaf's elements and the new element.
  - Values less than the median are put in the new left node and values greater than the median are put in the new right node, with the median acting as a separation value.
  - The separation value is inserted in the node's parent, which may cause it to be split, and so on. If the node has no parent (i.e., the node was the root), create a new root above this node (increasing the height of the tree).
B-Tree Queries

- Find the node with value 12
- Find nodes with value between 4 and 20.
A B+ tree is an n-ary tree with a variable but often large number of children per node. A B+ tree can be viewed as a B-tree in which each node contains only keys (not key-value pairs), and to which an additional level is added at the bottom with linked leaves. [Wikipedia]
B+Tree Insertion

- If the bucket is not full (at most $b - 1$ entries after the insertion), add the record.
- Otherwise, split the bucket.
  - Allocate new leaf and move half the bucket's elements to the new bucket.
  - Insert the new leaf's smallest key and address into the parent.
  - If the parent is full, split it too.
    - Add the middle key to the parent node.
  - Repeat until a parent is found that need not split.
- If the root splits, create a new root which has one key and two pointers. (That is, the value that gets pushed to the new root gets removed from the original node)
B+Tree Queries

- Find node with value 4
- Find nodes with value between 2 and 6.
● A Huffman code is a particular type of optimal prefix code that is commonly used for lossless data compression. The process of finding and/or using such a code proceeds by means of Huffman coding. [Wikipedia]
private static Node buildTrie(int[] freq)
{
    // Initialize priority queue with singleton trees.
    MinPQ<Node> pq = new MinPQ<Node>();
    for (char c = 0; c < R; c++)
        if (freq[c] > 0)
            pq.insert(new Node(c, freq[c], null, null));

    while (pq.size() > 1)
    {
        // Merge two smallest trees.
        Node x = pq.delMin();
        Node y = pq.delMin();
        Node parent = new Node('\0', x.freq + y.freq, x, y);
        pq.insert(parent);
    }

    return pq.delMin();
}
Example

- creating the Huffman code for some text (e.g. "she sells seashells")
- encoding the text using the Huffman code created
- decoding the encoded string