COL106: Data Structures and Algorithms

Ragesh Jaiswal, IIT Delhi

Definition ((2-4)-Tree)

A (2, 4)-Tree is a multiway search tree with the following two additional properties:

1. Size property: Every internal node has at most 4 children.
2. Depth property: All leaves have the same depth.

- Running time:
  - Search: $O(\log n)$
  - Insert: $O(\log n)$
  - Delete: $O(\log n)$

- We can easily generalise the techniques of (2, 4)-Tree to multiway search tree where instead of every internal node having at least 2 and at most 4 children to multiway search trees where every internal node have at least d and at most $2d$ children, where d is some constant.
- Such trees are known by the name B-tree and are used in modern filesystems and database implementations.

- AVL Tree and (2, 4)-Tree are just two examples of balanced search trees.
- There are many more examples of such trees.
- The book gives two other examples: red-black tree and Splay tree.

Data Structures: B-Tree

- Binary Search Tree (BST), AVL Tree, and $(2,4)$-tree are implementations of the Abstract Data Type called *Map* where key-value pairs with all distinct keys are stored and the primary supported operations are: get (search), put (insert), and remove (delete).
- All the above data structures and in fact all the data structures that we have seen (and implemented) in this class until now are memory-based. Meaning, that they are stored and accessed from primary memory.
- Suppose the data is so large that it does not make sense to implement a memory-based data structure and we have to design a disk-based data structure.
- For this, we will first have to understand how the disk is accessed.

- Suppose the data is so large that it does not make sense to implement a memory-based data structure and we have to design a disk-based data structure.
- For this, we will first have to understand how the disk is accessed:
  - There are slow mechanical operations involved when accessing data in a disk storage.
  - There is seek time for positioning the head at the correct place and transfer time for reading (or writing) data.
  - Disk access is performed in data chunks called blocks (typically size 4KB)
  - Disk access is significantly slower than memory access.

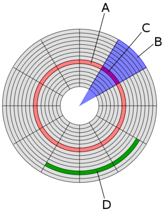


Figure : Tracks, Sectors, and Blocks on a disk.

- Suppose the data is so large that it does not make sense to implement a memory-based data structure and we have to design a disk-based data structure.
- For this, we will first have to understand how the disk is accessed:
  - There are slow mechanical operations involved when accessing data in a disk storage.
  - There is seek time for positioning the head at the correct place and transfer time for reading (or writing) data.
  - Disk access is performed in data chunks called blocks (typically size 4KB)
  - Disk access is significantly slower than memory access.
- Question: Are BST, AVL-Tree, (2, 4)-tree appropriate for disk-based implementation?

- Suppose the data is so large that it does not make sense to implement a memory-based data structure and we have to design a disk-based data structure.
- For this, we will first have to understand how the disk is accessed:
  - There are slow mechanical operations involved when accessing data in a disk storage.
  - There is seek time for positioning the head at the correct place and transfer time for reading (or writing) data.
  - Disk access is performed in data chunks called blocks (typically size 4KB)
  - Disk access is significantly slower than memory access.
- Question: Are BST, AVL-Tree, (2, 4)-tree appropriate for disk-based implementation?
- Goals of disk-based implementation:
  - Space usage should be linear in the size of the data.
  - The number of disk accesses should be as small as possible.

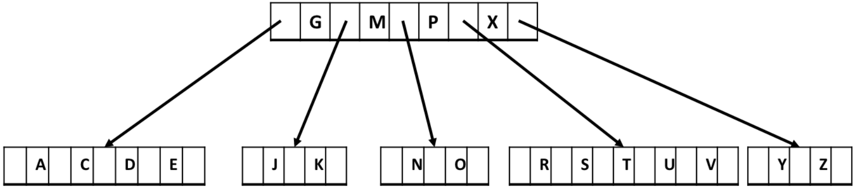
- Goals of disk-based implementation:
  - Space usage should be linear in the size of the data.
  - The number of disk accesses should be as small as possible.
- Suppose a disk block can store m key-value pairs (in addition to $m + 1$ pointers).
- Can you think of a data structure that will be appropriate in this context?

- Goals of disk-based implementation:
  - Space usage should be linear in the size of the data.
  - The number of disk accesses should be as small as possible.
- Suppose a disk block can store m key-value pairs (in addition to $m+1$ pointers).
- Can you think of a data structure that will be appropriate in this context?
- Consider $(d, 2d)$-Tree which is a generalisation of $(2, 4)$-tree where each internal node should hold at least $(d-1)$ entries (*except root*) and at most $(2d-1)$ entries. We can generalise all operations studied for $(2, 4)$-tree.
  - Note that $d = 2$ for $(2, 4)$-tree.
- What is the height h of a $(d, 2d)$-tree containing n entries?

- Goals of disk-based implementation:
  - Space usage should be linear in the size of the data.
  - The number of disk accesses should be as small as possible.
- Suppose a disk block can store m key-value pairs (in addition to $m+1$ pointers).
- Can you think of a data structure that will be appropriate in this context?
- Consider $(d, 2d)$-Tree which is a generalisation of $(2, 4)$-tree where each internal node should hold at least $(d-1)$ entries (*except root*) and at most $(2d-1)$ entries. We can generalise all operations studied for $(2, 4)$-tree.
  - Note that $d = 2$ for $(2, 4)$-tree.
- What is the height h of a $(d, 2d)$-tree containing n entries? $h = O(\log_d n)$
- What is the value of d we should use in the current context?

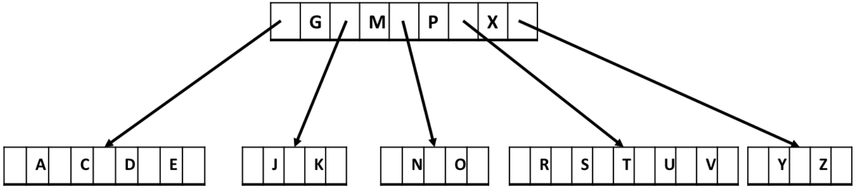
- Goals of disk-based implementation:
  - Space usage should be linear in the size of the data.
  - The number of disk accesses should be as small as possible.
- Suppose a disk block can store m key-value pairs (in addition to $m+1$ pointers).
- Can you think of a data structure that will be appropriate in this context?
- Consider $(d, 2d)$-Tree which is a generalisation of $(2, 4)$-tree where each internal node should hold at least $(d-1)$ entries (*except root*) and at most $(2d-1)$ entries. We can generalise all operations studied for $(2, 4)$-tree.
  - Note that $d = 2$ for $(2, 4)$-tree.
- What is the height h of a $(d, 2d)$-tree containing n entries? $h = O(\log_d n)$
- What is the value of d we should use in the current context? $\frac{m+1}{2}$
- Typical value of $m \approx 1000$. What is the minimum number of keys a B-Tree of height 2 can store?

- Let us consider an example of B-Tree where $m = 5$ (so $d = 3$)



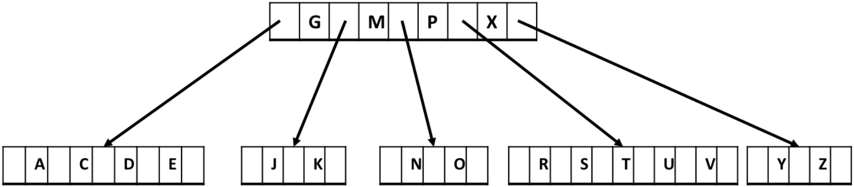
- Insert the following keys (in that order): B, Q, L , F.

- Let us consider an example of B-Tree where $m = 5$ (so $d = 3$)



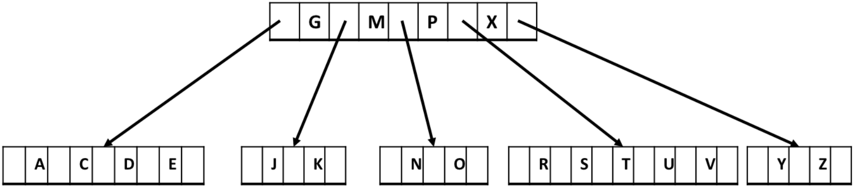
- Insert the following keys (in that order): B, Q, L , F.
- What is the bound on the number of disk accesses for an insert operation?

- Let us consider an example of B-Tree where $m = 5$ (so $d = 3$)



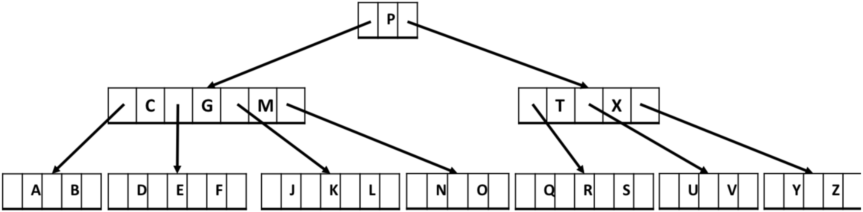
- Insert the following keys (in that order): B, Q, L , F.
- What is the bound on the number of disk accesses for an insert operation? $O(\log_d n)$
- What is the CPU-time?

- Let us consider an example of B-Tree where $m = 5$ (so $d = 3$)



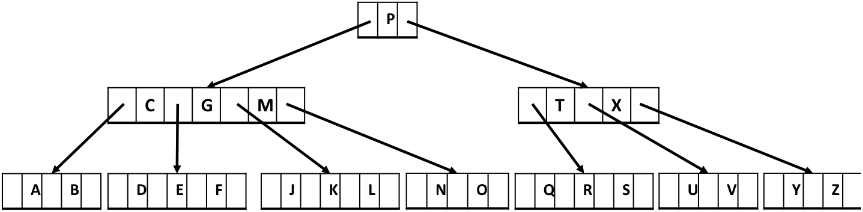
- Insert the following keys (in that order): B, Q, L , F.
- What is the bound on the number of disk accesses for an insert operation? $O(\log_d n)$
- What is the CPU-time? $O(d \log_d n)$

- Let us consider an example of B-Tree where $m = 5$ (so $d = 3$)



- Delete the following keys (in that order): F, M, G

- Let us consider an example of B-Tree where $m = 5$ (so $d = 3$)



- Delete the following keys (in that order): F, M, G
- What is the bound on the number of disk accesses for a delete operation? $O(\log_d n)$
- What is the CPU-time? $O(d \log_d n)$

Definition ((2-4)-Tree)

A (2, 4)-Tree is a multiway search tree with the following two additional properties:

1. Size property: Every internal node has at most 4 children.
2. Depth property: All leaves have the same depth.

- Running time:
  - Search: $O(\log n)$
  - Insert: $O(\log n)$
  - Delete: $O(\log n)$
- Exercise 1: Insert the following sequence of keys into an initially empty $(2, 4)$-tree: $5, 16, 22, 45, 2, 10, 18, 30, 50, 12, 1$

Definition ((2-4)-Tree)

A (2, 4)-Tree is a multiway search tree with the following two additional properties:

1. Size property: Every internal node has at most 4 children.
2. Depth property: All leaves have the same depth.

- Running time:
  - Search: $O(\log n)$
  - Insert: $O(\log n)$
  - Delete: $O(\log n)$
- Exercise 1: Insert the following sequence of keys into an initially empty (2, 4)-tree: $5, 16, 22, 45, 2, 10, 18, 30, 50, 12, 1$.
- Exercise 2: Delete keys $45, 18, 12$ (in that order) from the tree obtained in Exercise 1.

End