Name : Kreena Shah
Sapid : 60004210243
Batch : Comps C’32
Subject : Advance Database Management System (ADBMS)

Experiment No 4

Aim : Optimization Using B & B+ Trees

Write Ups :

Exp 4 : Optimize Using B & B+ Trees 13/15

Name : Kareena Shah
Sapid : 60004210243
Batch : Comps C'32
Subject : ADBMS
Date of Performance : 26/10/2023
Date of Submission : 02/11/2023

Aim : Optimize using B & B+ Trees

Theory :

B-Tree

B-Tree is a self leaning, balancing, tree data structure that plays a crucial role in advanced database management systems. It is designed to make a balance between read & write operations making it efficient for indexing & searching in database systems.

Features :

(1) Balanced Structure

B-trees are balanced, meaning all leaf nodes are at the same level. This balance ensures that search & insertion operations remain consistent & efficient.

(2) Ordered Data

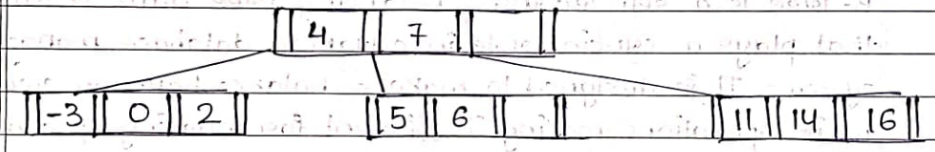
B-Trees organize data in a sorted manner, which is particularly useful for range queries & data retrieval operations.

(3) Split & Merge

When a B-Tree node becomes full during an insertion operation, it splits into two nodes

(4) Fast Search
Provides efficient search operations, often requiring only a few comparison to locate a specific item within large dataset

(5) Node Capacity
The no. of children a node can have is determined by the order of the B-tree. Higher order B-trees reduce the trees height & improve performance



B+ Trees

B+ Trees is an extension of B-Tree data structure with a focus on optimizing range queries & disk storage in database systems. In ADBMS, B+ trees offer several advantages for handling data efficiently

Features :

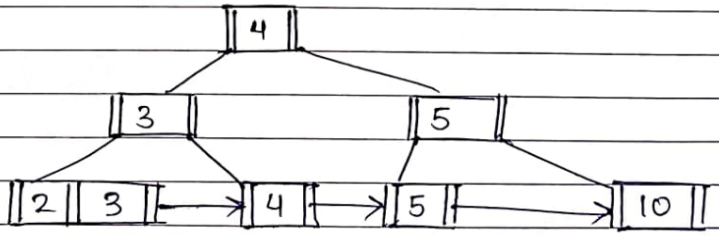
(1) Balanced & Ordered
Like B-Trees, balanced & ordered data is maintained, ensuring efficient insertion, deletion & search operations.

(2) Reduced Height
This have higher branching factor & Keeps the tree height shallow, leading to quicker access time

(3) Non Leaf Nodes
They store key values & add as an index to guide searches in tree

(4) Leaf Nodes
All data is stored in the leaf nodes, which are linked together in a doubly linked list. This structure is advantageous for range queries, as scanning data becomes more straightforward.

(5) Disk Based Storage
B+ Trees are well united for database systems because they optimize disk storage. Sequential access of operations are efficient, reducing disk I/O operations.



Search Time & Insertion Time

For 10000000 records,

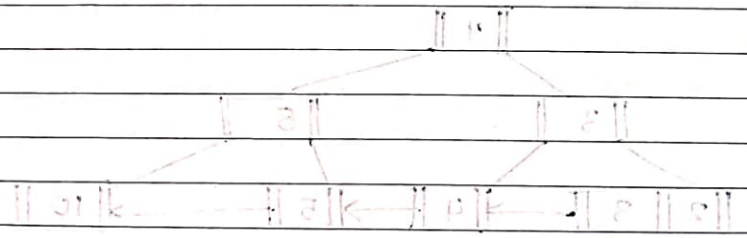
For 'Key' = 33333

Search Time for B Tree = 5437.129 millisecond

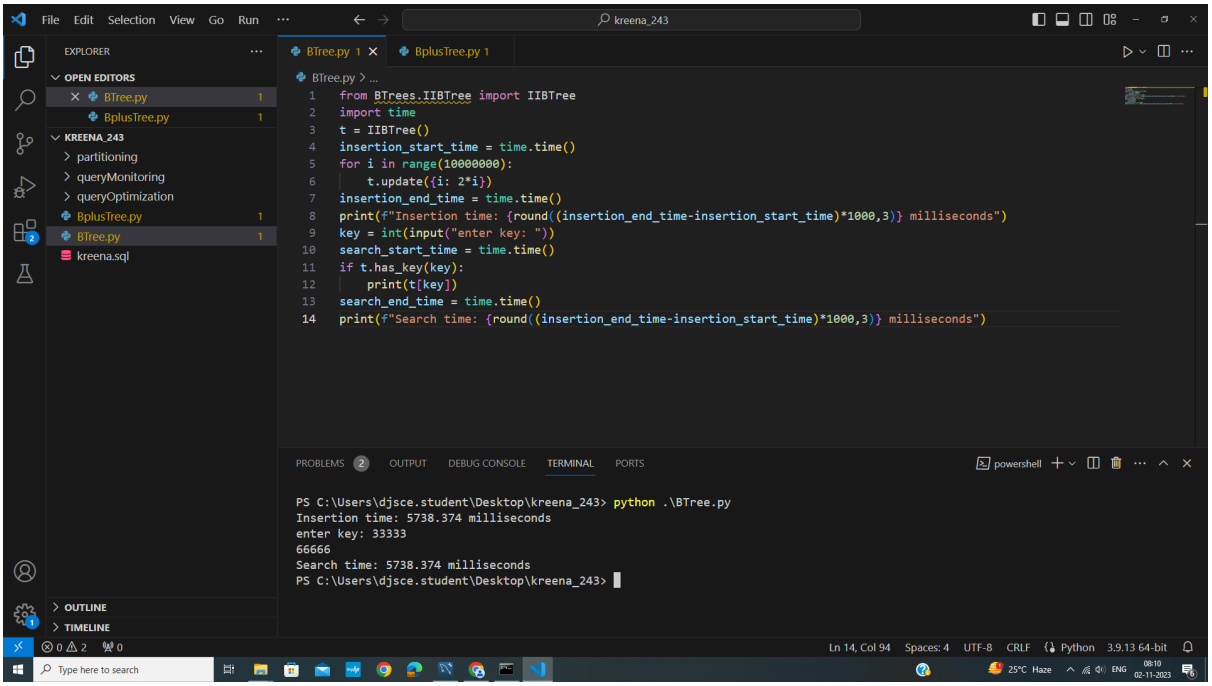
B+ Tree = 5613.673 millisecond

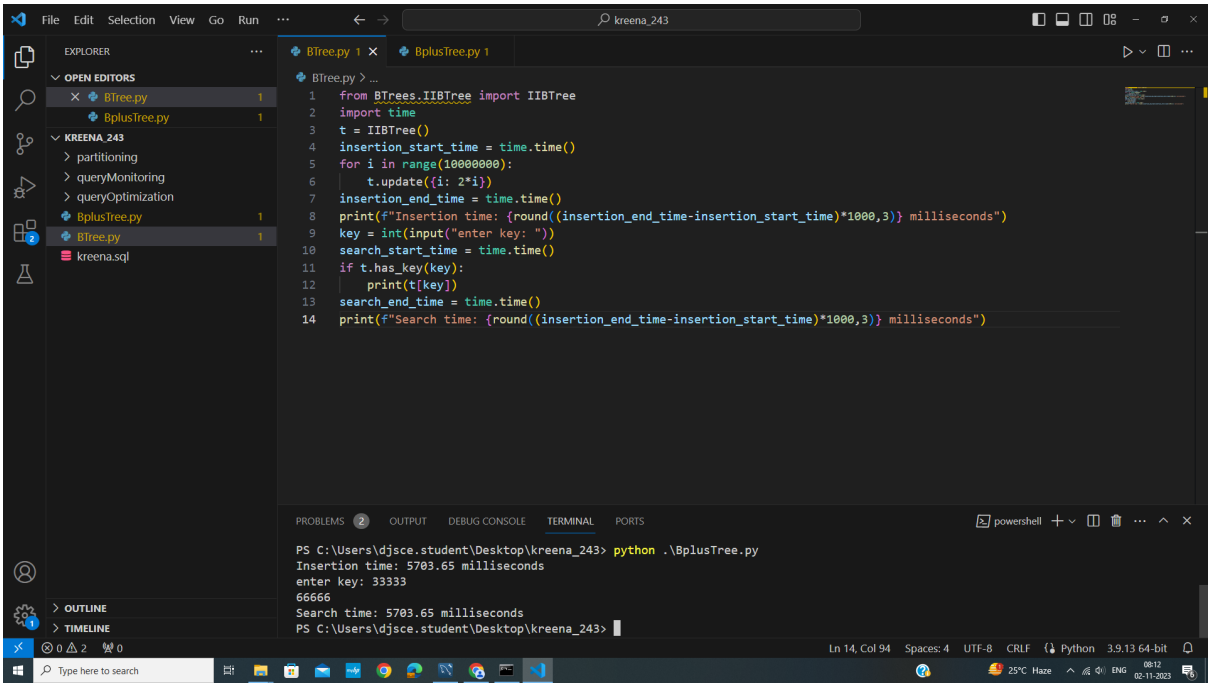
Conclusion :

Optimization using B & B+ Tree aims to reduce overhead & improve efficiency of data operations resulting in faster & more streamlined data management



Screenshots :


File Edit Selection View Go Run
kreena_243
EXPLORER
OPEN EDITORS
BTree.py 1
BplusTree.py 1
KREENA_243
partitioning
queryMonitoring
queryOptimization
BplusTree.py 1
BTree.py 1
kreena.sql
OUTLINE
TIMELINE
BTree.py 1
BplusTree.py 1
BTree.py > ...
1 from BTrees.IIBTree import IIBTree
2 import time
3 t = IIBTree()
4 insertion_start_time = time.time()
5 for i in range(10000000):
6 t.update({i: 2*i})
7 insertion_end_time = time.time()
8 print(f"Insertion time: {round((insertion_end_time-insertion_start_time)*1000,3)} milliseconds")
9 key = int(input("enter key: "))
10 search_start_time = time.time()
11 if t.has_key(key):
12 print(t[key])
13 search_end_time = time.time()
14 print(f"Search time: {round((insertion_end_time-insertion_start_time)*1000,3)} milliseconds")
PROBLEMS 2 OUTPUT DEBUG CONSOLE TERMINAL PORTS
powershell
PS C:\Users\djsce.student\Desktop\kreena_243> python .\BTree.py
Insertion time: 5738.374 milliseconds
enter key: 33333
66666
Search time: 5738.374 milliseconds
PS C:\Users\djsce.student\Desktop\kreena_243>
Ln 14, Col 94 Spaces: 4 UTF-8 CRLF Python 3.9.13 64-bit
Type here to search
25°C Haze
ENG
08:10
02-11-2023


File Edit Selection View Go Run
kreena_243
EXPLORER
OPEN EDITORS
BTree.py 1
BplusTree.py 1
KREENA_243
partitioning
queryMonitoring
queryOptimization
BplusTree.py 1
BTree.py 1
kreena.sql
OUTLINE
TIMELINE
BTree.py 1
BplusTree.py 1
BTree.py > ...
1 from BTrees.IIBTree import IIBTree
2 import time
3 t = IIBTree()
4 insertion_start_time = time.time()
5 for i in range(10000000):
6 t.update({i: 2*i})
7 insertion_end_time = time.time()
8 print(f"Insertion time: {round((insertion_end_time-insertion_start_time)*1000,3)} milliseconds")
9 key = int(input("enter key: "))
10 search_start_time = time.time()
11 if t.has_key(key):
12 print(t[key])
13 search_end_time = time.time()
14 print(f"Search time: {round((insertion_end_time-insertion_start_time)*1000,3)} milliseconds")
PROBLEMS 2 OUTPUT DEBUG CONSOLE TERMINAL PORTS
powershell
PS C:\Users\djsce.student\Desktop\kreena_243> python .\BplusTree.py
Insertion time: 5703.65 milliseconds
enter key: 33333
66666
Search time: 5703.65 milliseconds
PS C:\Users\djsce.student\Desktop\kreena_243>
Ln 14, Col 94 Spaces: 4 UTF-8 CRLF Python 3.9.13 64-bit
Type here to search
25°C Haze
ENG
08:12
02-11-2023