# Data Structures and Algorithms

# CSYE 6205

# Homework 7

# Due: April 4, 2022

Put all your java, compiled class files and documentation into zip file named Homework6.zip and submit it via canvas before the END of due date. Put your name on all .java files.

1. Consider a disk with a sector size of 512 bytes, 1,000 tracks per surface, 100 sectors per track, five double-sided platters and a block/page size of 1024 bytes. Suppose that the average seek time is 4 msec, the average rotational delay is 4 msec, and the transfer rate is 100 MB per second. Suppose that a file containing 1,000,000 records of 400 bytes each to be stored on disk and that no record is allowed to span two blocks.

a) How many records fit onto a block?

b) How many blocks are required to store the entire file?

c) What are the design decisions in constructing BTree? What are the relationships

between BTree Data structure, Memory, Disk, and CPU? Explain

2. Consider data: {45,1,5,14, 24,13,11,8,19,4,31,35,56,17,29,6, 33}

a) Construct a B-Tree order of m=5

b) Delete elements 45, 19, 24, Construct the new BTree, and Describe

the delete algorithm step(s) for each element.

c) minimum-degree, minKeys, maxKeys, minChildren, maxChildren?

d) Write Java code for (a) and (b)

e) Perform Inorder, Postorder, Preorder Traversals on B-Tree

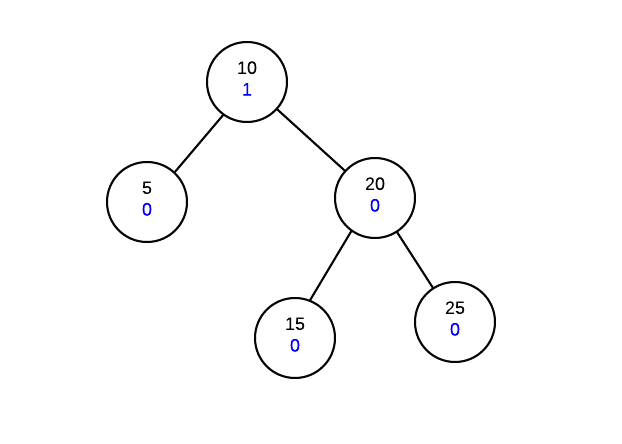
3. Consider data: 9, 18, 5, 15, 17, 26, 39, 75

a) Construct Red-Black Tree

b) In constructing tree, why Hight and Balance are important factors?

c) What is time-complexity of Red-Black tree?

4. Consider the following AVL Tree:



AVL trees work by ensuring that the tree is height balanced after an operation. Each node

must not only maintain its data and children information, but also a height balance value.

The height balance of a node is calculated as follows:

Height-balance-of-node = height of right-subtree - height of left-subtree

The above formula means that if the right-subtree is taller, the height balance of the node will

be positive. If the left-subtree is taller, the balance of the node will be negative.

Show AVL Tree after performing these insert operations:

Insert 28

Insert 39

Insert 35

The height of AVL Tree is not balanced after insertions. You would need to do several

rotations, single-rotation and double-rotation to make tree balanced.

a) Show all rotations to make tree height balanced

b) Write AVL Tree Algorithm

c) Write Java code for algorithm and Run

d) What is time-complexity of AVL algorithm?

5. Consider the following data:

a) Draw the Undirected graph that is represented as follows:

Vertices: 1, 2, 3, 4, 5, 6, 7, 8, 9

Edges:  (5, 7), (4, 6), (3, 7), (3, 9), (6, 7), (5, 7), (1, 4), (2, 4), (2, 3), (4, 7), (4, 8)

b) Is graph connected? Is it complete? Explain.

c) Draw Directed graph using the same data as above.

d) Is the directed graph connected? Is it complete? Explain.

e) Write Adjacency lists representation for data in (a)

f) Write Java code implementation of Adjacency lists representation.

Note: you need to use Bag data structure.

g) What is the space and running time complexity of Adjacency Lists algorithm?

h) build Adjacency Matrix for data in (a)

i) For graph in (a**)** perform:

i) Depth-First Search traversal algorithm starting at vertex 1

ii) Write Java code for the algorithm

iii) What is the running time complexity of the algorithm?

j) For graph in (a) perform:

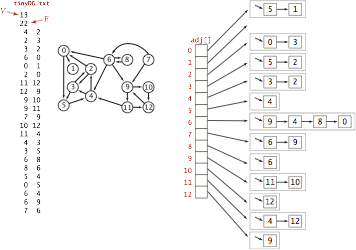
i) Breadth-First Search traversal starting at Vertex 1.

ii) Write Java code for the algorithm.

iii) What is the running time complexity of the algorithm?

6. Consider Graph data structures for Adjacency-Lists for both Directed graph

and Undirected Graph, What are the differences (data structures, methods)?

7. Consider this Directed graph:

a) Construct Adjacency List for Graph

b) Write Java code to iterate through Graph and print connection for each Vertex

8. Consider the following sequence of letters:

‘A’G’F’B’K’D’H’M’J'E'S'I'R'X'C'L'N'T'U'P'

a) Build BTree with order of t=4

b) What is minimum degree for this BTree?

c) Write Java code to Insert into BTree

d) Consider 3-cases for deleting from B-tree,

Delete M, F, E

e) Write Java code for all 3 deletion cases

f) Test Java code for (c) and (e) for BTree you constructed in (a)

g) Discuss height, time and space complexity

APPENDIXES

SEARCHING B-Tree

1: function B-Tree-Search(Node x, Key k)

2: i ← 0

3: while i < numkeys(x) and k > x.keys[i] do

4: i ← i + 1

5: end while

6: if i < numkeys(x) and k = x.keys[i] then return (x, i)

7: end if

8: if leaf(x) then

9: return (x, NULL)

10: else

11: return B-Tree-Search(x.child[i], k)

12: end if

13: end function

INSERTING into B-Tree

1: procedure B-Tree-Insert(Node x, Key k)

2: find i such that x.keys[i] > k or i >=numkeys(x)

3: if x is a leaf then

4: Insert k into x.keys at i

5: else

6: if x.child[i] is full then

7: Split x.child[i]

8: if k > x.key[i] then

9: i ← i + 1

10: end if

11: end if

12: B-Tree-Insert(x.child[i], k)

13: end if

14: end procedure

DELETING from B-Tree

1. x is a leaf and contains the key (it will have at least t keys). This is

trivial - just delete the key.

2. x is an internal node and contains the key. There are 3 subcases:

2a: predecessor child node has at least t keys

2b: successor child node has at least t keys

2c: neither predecessor nor successor child has t keys

3. x is an internal node, but doesn’t contain the key. Find the child subtree

of x that contains the key if it exists (call the child c). There are three subcases:

3a: child c has at least t keys. Simply recurse to c.

3b: child c has t − 1 keys and one of its siblings has t keys.

3c: child c and both siblings have t − 1 keys.