Data Structures and Algorithms

INFO 6205

Homework 7

Due: Sunday 26, 2019

Name: Qian Cai

NU ID:001389278

Put all your work, code, compiled, and executable files and documentation into a zip file named Homework7.zip and submit it via the drop box on the blackboard by the END of due date. Put your name on all code files. There will be a short quiz on this homework.

1. a) Draw the undirected graph that is represented as follows:

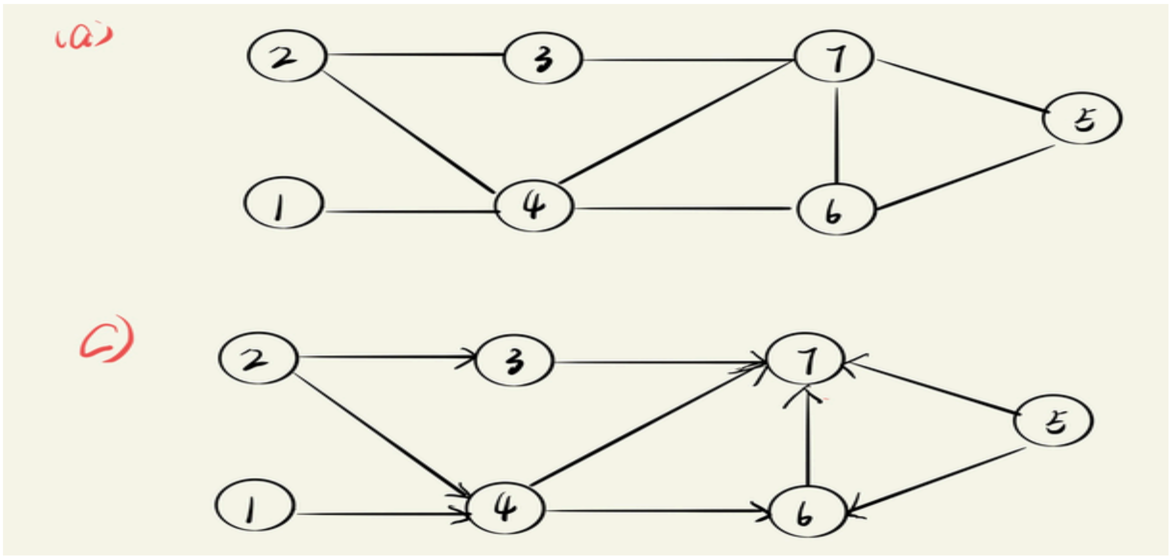
   Vertices: 1, 2, 3, 4, 5, 6, 7

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

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

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

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



b) It is connected because you can reach another from every node by the possible routes.

It is incomplete because the complete undirected graph is a graph in which every pair of distinct vertices is connected by a unique edge. In this graph, every pair of distinct vertices is not connected by a unique edge. For example, the vertice 1 is not connected by a unique edge with the vertices except 4 in this graph.

d)It is unconnected because the connected directed graph in which it is possible to reach any node starting from any other node by traversing edges in some direction. However, it is impossible to reach any node starting from any other node by traversing edges in some direction in this graph. For example, vertice 2 is unable to reach 1.

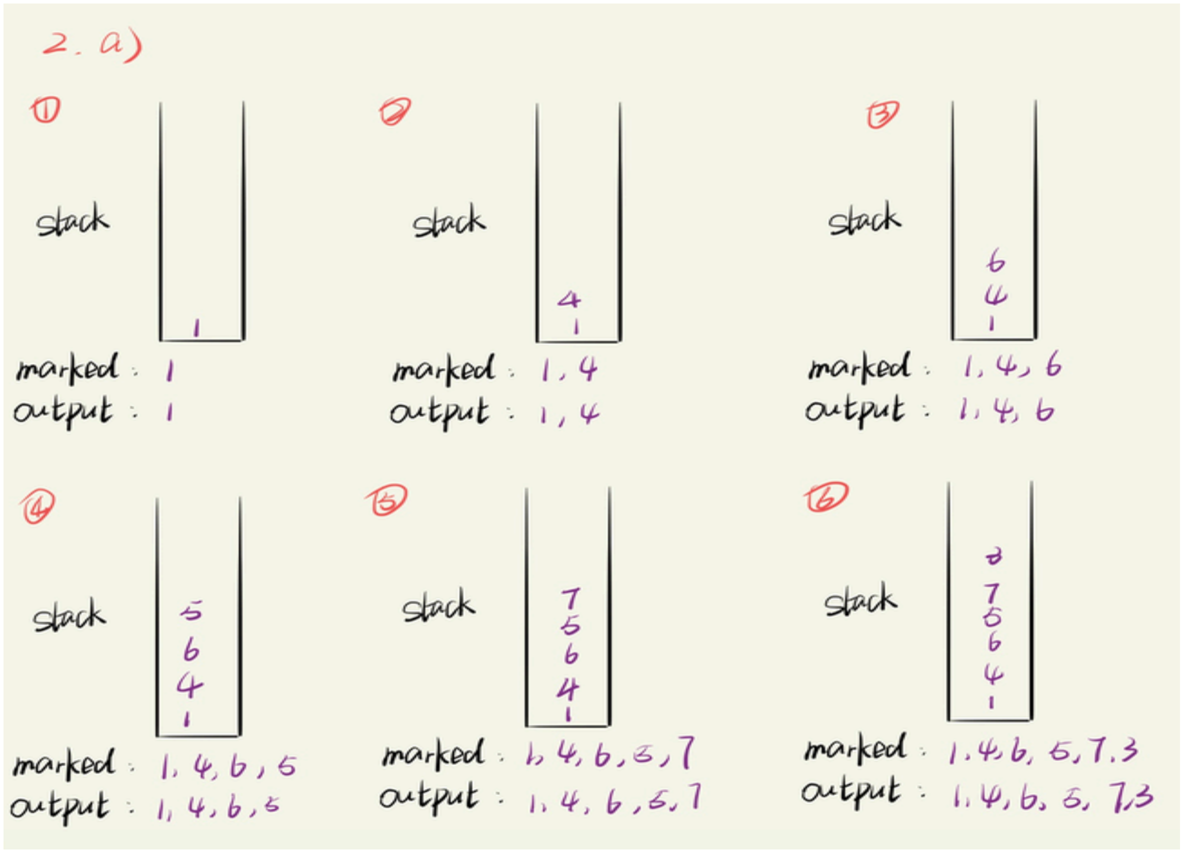
It is incomplete because the complete directed graph is a graph in which every pair of distinct vertices is connected by a pair of unique edges. In this graph, every pair of distinct vertices is not connected by a pair of unique edges. For example, the vertice 1 is not connected by a pair of unique edges with all vertices in this graph。

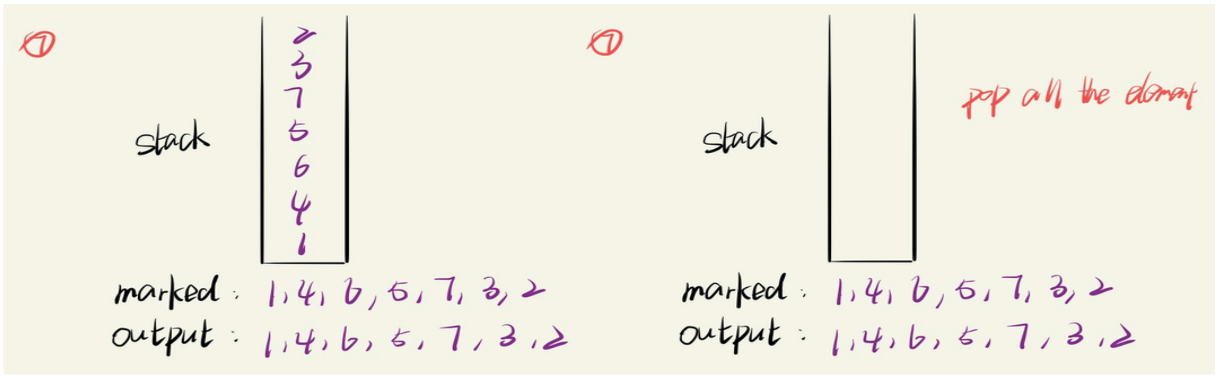
2. For graph **1.a** perform:

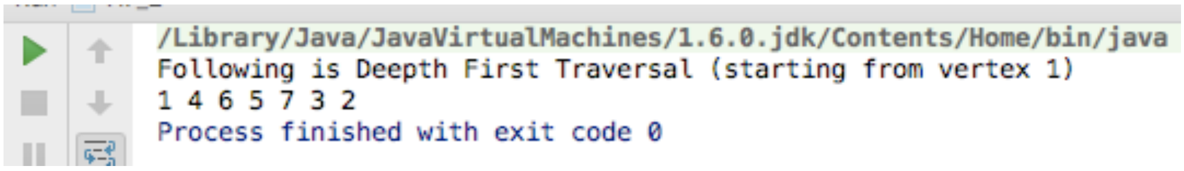
   a) Depth-First Search traversal algorithm starting at vertex 1

   b) Write Java code for the algorithm

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

a) 


⑦
stack
2
3
7
5
6
4
1
marked : 1, 4, 6, 5, 7, 3, 2
output : 1, 4, 6, 5, 7, 3, 2
⑦
stack
pop all the element
marked : 1, 4, 6, 5, 7, 3, 2
output : 1, 4, 6, 5, 7, 3, 2

b) 

```
/Library/Java/JavaVirtualMachines/1.6.0.jdk/Contents/Home/bin/java
Following is Deepth First Traversal (starting from vertex 1)
1 4 6 5 7 3 2
Process finished with exit code 0
```

The code is in H7_2.java

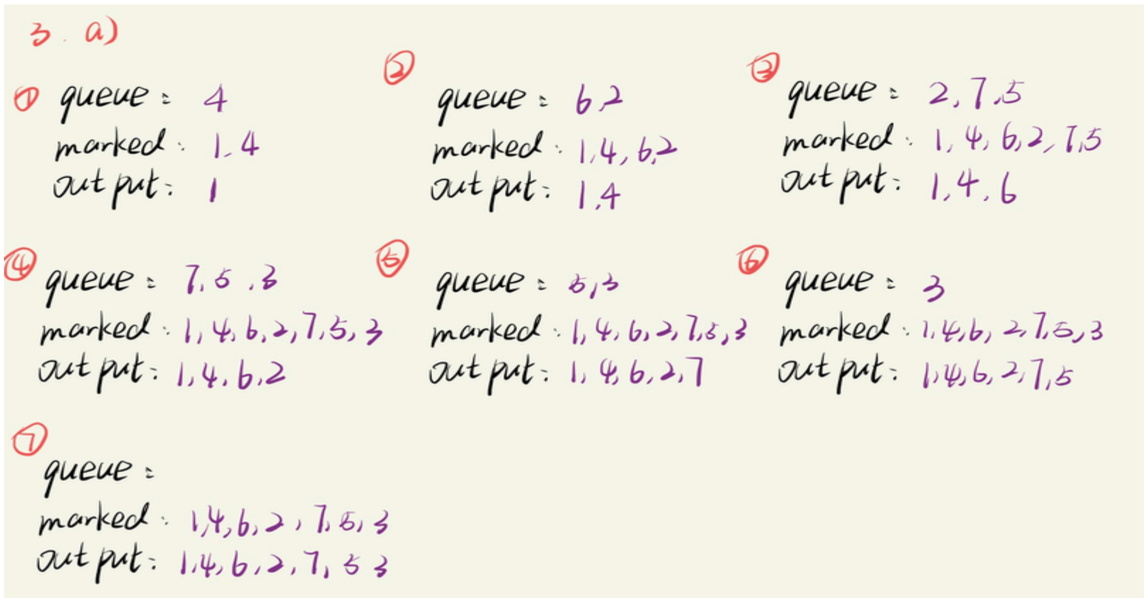
c) Time Complexity: O(V+E) where V is number of vertices in the graph and E is number of edges in the graph

3. For graph **1.a** perform:

   a) Breadth-First Search traversal starting at Vertex 1

   b) Write Java code for the algorithm.

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



3. a)

① queue: 4
marked: 1, 4
output: 1

② queue: 6, 2
marked: 1, 4, 6, 2
output: 1, 4

③ queue: 2, 7, 5
marked: 1, 4, 6, 2, 7, 5
output: 1, 4, 6

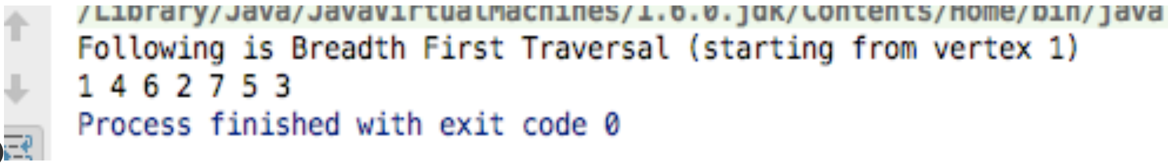
④ queue: 7, 5, 3
marked: 1, 4, 6, 2, 7, 5, 3
output: 1, 4, 6, 2

⑤ queue: 5, 3
marked: 1, 4, 6, 2, 7, 5, 3
output: 1, 4, 6, 2, 7

⑥ queue: 3
marked: 1, 4, 6, 2, 7, 5, 3
output: 1, 4, 6, 2, 7, 5

⑦ queue:
marked: 1, 4, 6, 2, 7, 5, 3
output: 1, 4, 6, 2, 7, 5 3

a)



```
/Library/Java/JavaVirtualMachines/1.6.0.jdk/Contents/Home/bin/java
Following is Breadth First Traversal (starting from vertex 1)
1 4 6 2 7 5 3
Process finished with exit code 0
```

b)

The code is in H7_3.java

c) Time Complexity: O(V+E) where V is number of vertices in the graph and E is number of edges in the graph

4. a) Consider Graph data structures for Adjacency-Lists for both directed graph and undirected Graph

   b) What are the differences (data structures, methods)

a) The data structures for Adjacency-Lists is linkedlist

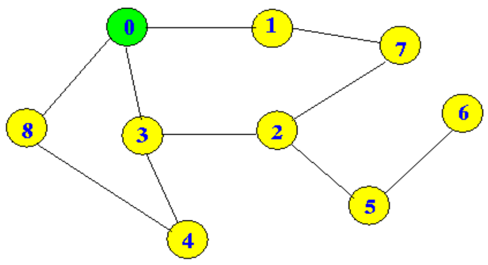
b) The "addEdge" method of directed graph need to add "indegree[w]++" to mark the amount of indegree of each vertice and "adj[v].add(w);"

The "addEdge" method of undirected graph do not need to add "indegree[w]++" to mark the amount of indegree of each vertice and "adj[v].add(w); adj[w].add(v);"

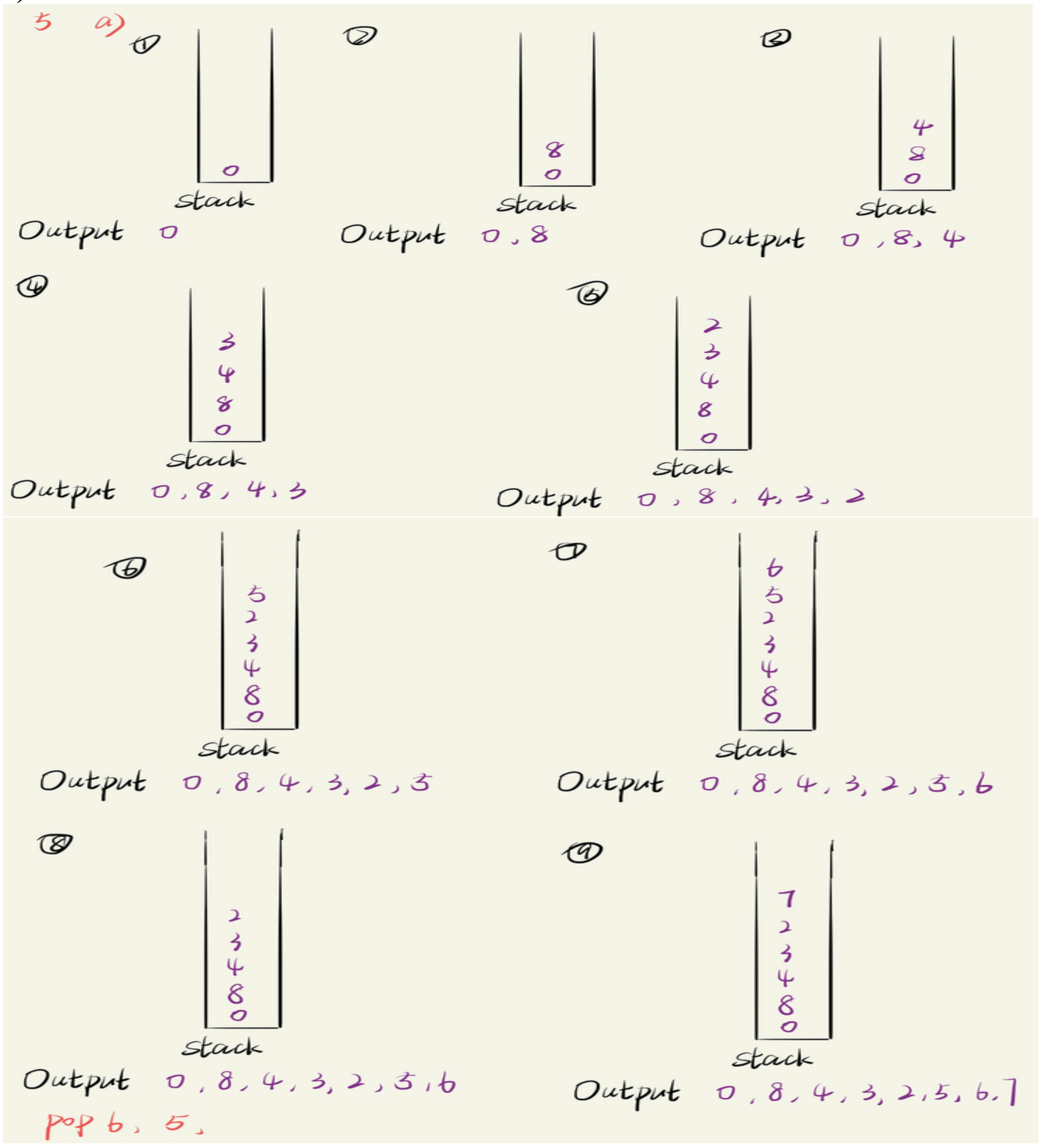
5. Consider the following Graph perform:

   a) Depth-First Search, Show Data structure step-by-step,

   b) Breadth-First Search show step-by-step



a)



Output 0

Output 0, 8

Output 0, 8, 4

Output 0, 8, 4, 3

Output 0, 8, 4, 3, 2

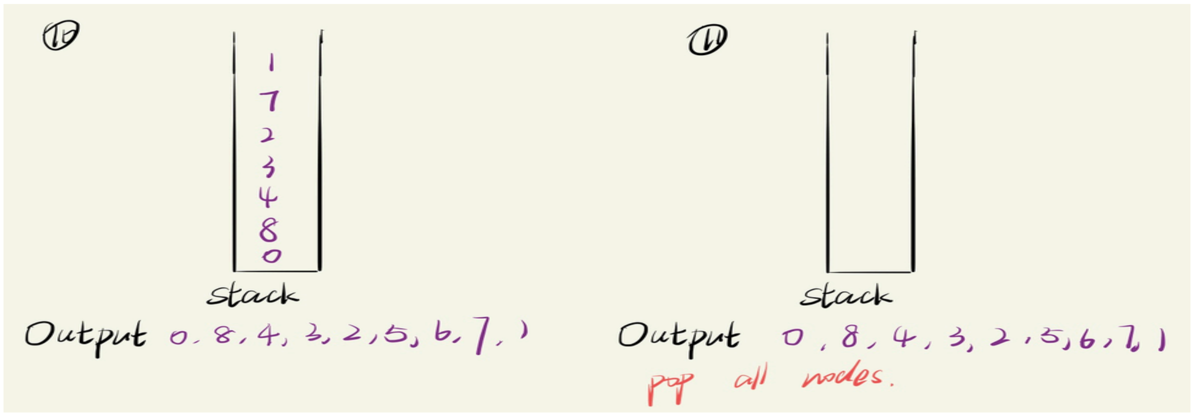
Output 0, 8, 4, 3, 2, 5

Output 0, 8, 4, 3, 2, 5, 6

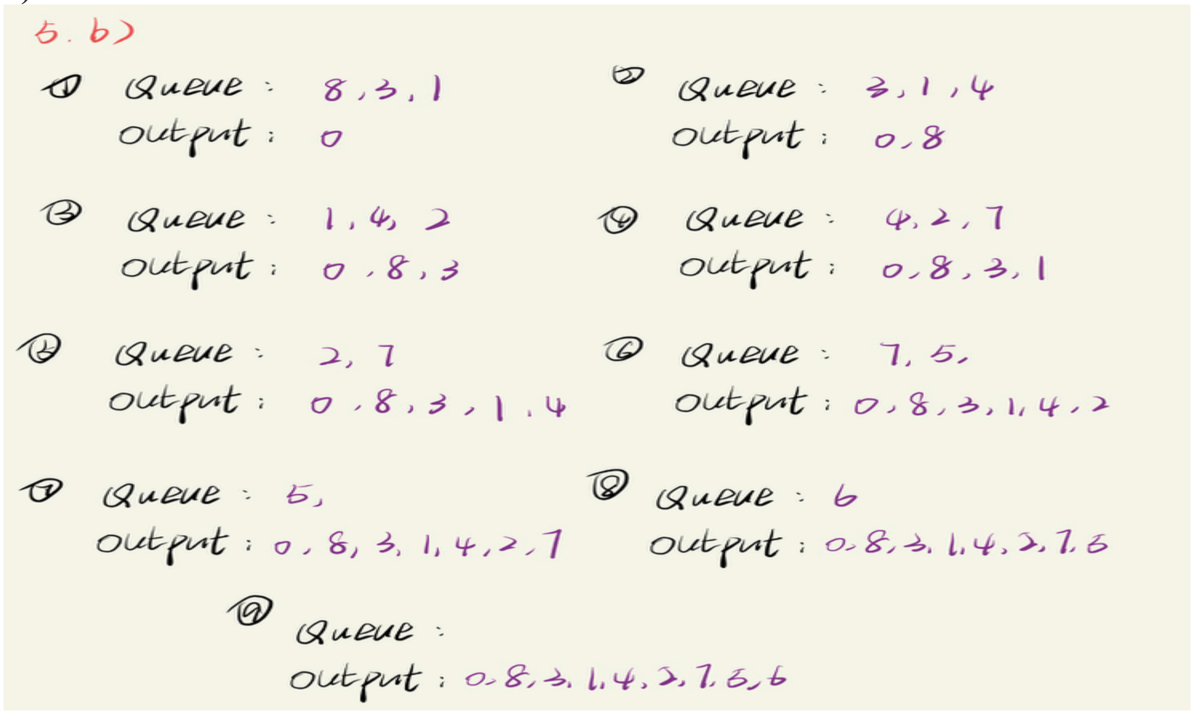
Output 0, 8, 4, 3, 2, 5, 6

pop 6, 5,

Output 0, 8, 4, 3, 2, 5, 6, 7


⑪
1
7
2
3
4
8
0
stack
Output 0, 8, 4, 3, 2, 5, 6, 7, 1
⑫
stack
Output 0, 8, 4, 3, 2, 5, 6, 7, 1
pop all nodes.

b)



5. b)

① Queue : 8, 3, 1
output : 0

② Queue : 3, 1, 4
output : 0, 8

③ Queue : 1, 4, 2
output : 0, 8, 3

④ Queue : 4, 2, 7
output : 0, 8, 3, 1

⑤ Queue : 2, 7
output : 0, 8, 3, 1, 4

⑥ Queue : 7, 5,
output : 0, 8, 3, 1, 4, 2

⑦ Queue : 5,
output : 0, 8, 3, 1, 4, 2, 7

⑧ Queue : 6
output : 0, 8, 3, 1, 4, 2, 7, 5

⑨ Queue :
output : 0, 8, 3, 1, 4, 2, 7, 5, 6

6. a) Construct a B-Tree order of t=4 for data: {3,7,9,23,45,1,5,14,5,24,13,11,8,19,4,31,35,56}

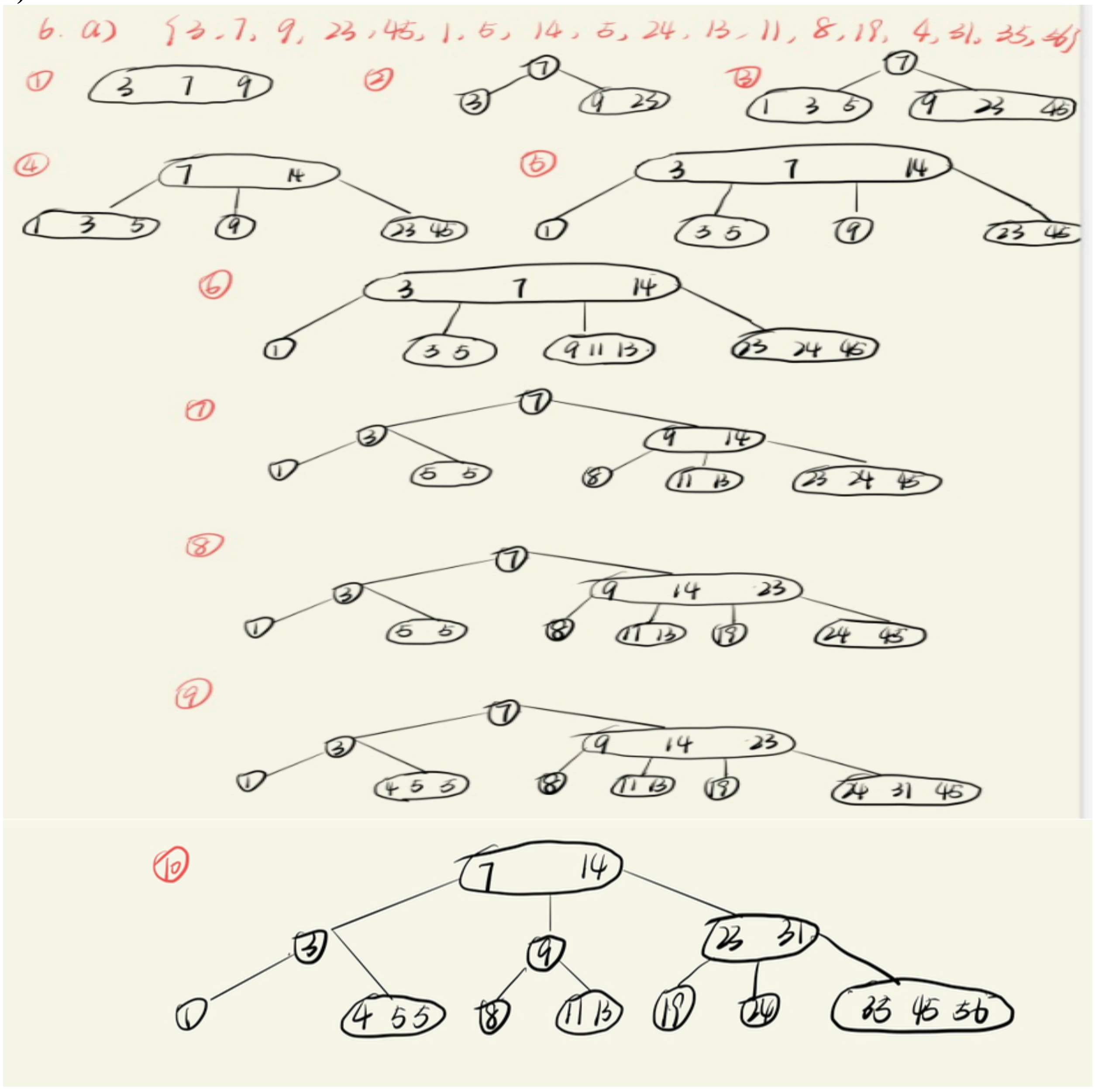
b) Delete elements 45, 11, 31, Construct the new BTree, and Describe the delete algorithm step(s) for each element.

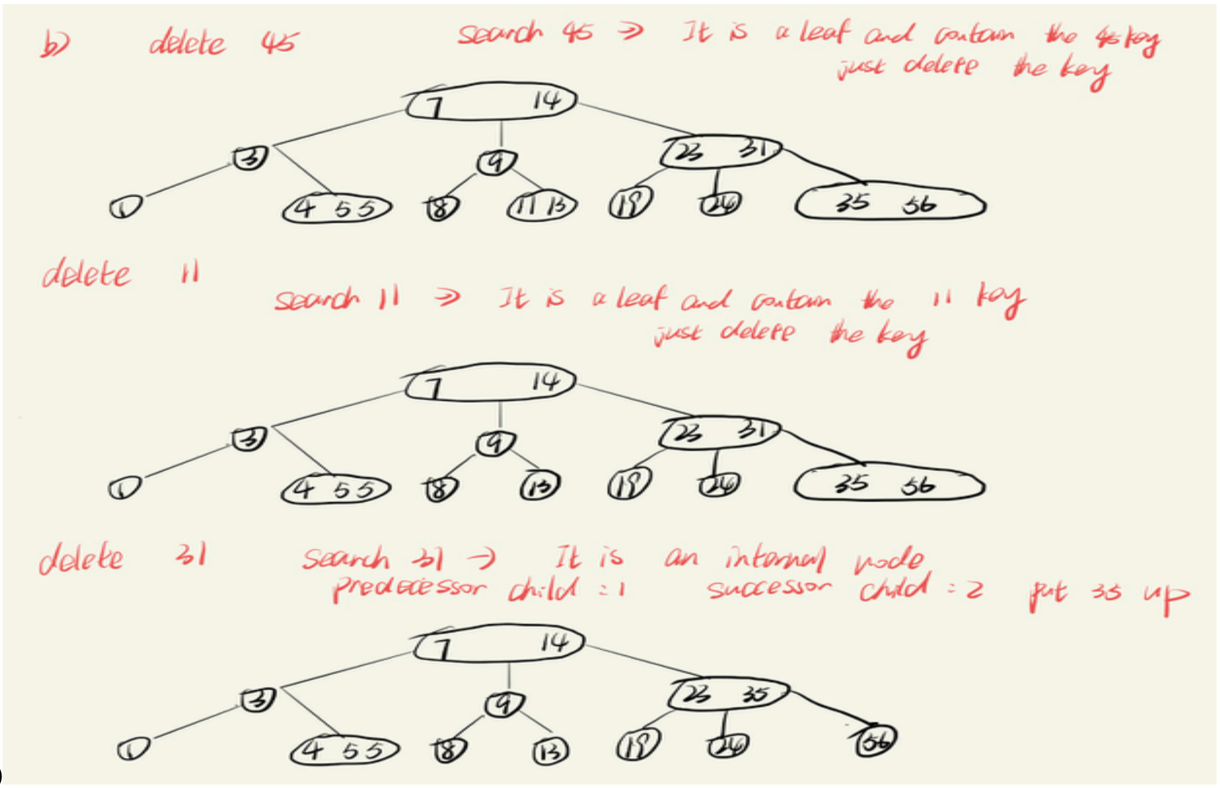
c) class Record is described below. Write Java code to build the b-tree you constructed in (a)

```
public class Record {
    private int key
    private Node leftNode;
    private Node rightNode;
    public Record(int key, Node leftNode, Node rightNode) {
        this.key = key;
        this.leftNode = leftNode;
        this.rightNode = rightNode; }
    public Record(int key){
        this.key = key;  }
    public int getKey() {
```

```
    return key; }
public Node getLeftNode() {
    return leftNode; }
public Node getRightNode() {
    return rightNode;  }
public void setKey(int key) {
    this.key = key;}
public void setLeftNode(Node leftNode) {
    this.leftNode = leftNode; }
public void setRightNode(Node rightNode) {
    this.rightNode = rightNode;}}
```

a)


6. a) {3, 7, 9, 23, 45, 1, 5, 14, 5, 24, 13, 11, 8, 18, 4, 31, 35, 56}
①
②
③
④
⑤
⑥
⑦
⑧
⑨
⑩


b) delete 45
search 45 ⇒ It is a leaf and contain the 45 key just delete the key
delete 11
search 11 ⇒ It is a leaf and contain the 11 key just delete the key
delete 31
search 31 → It is an internal node
predecessor child : 1 successor child : 2 put 35 up

b)

c) The code is in H7_6

7. Consider Data: {9 30 -28 -25 -2 -3 6 12 15 19 25 26 33 36 39}

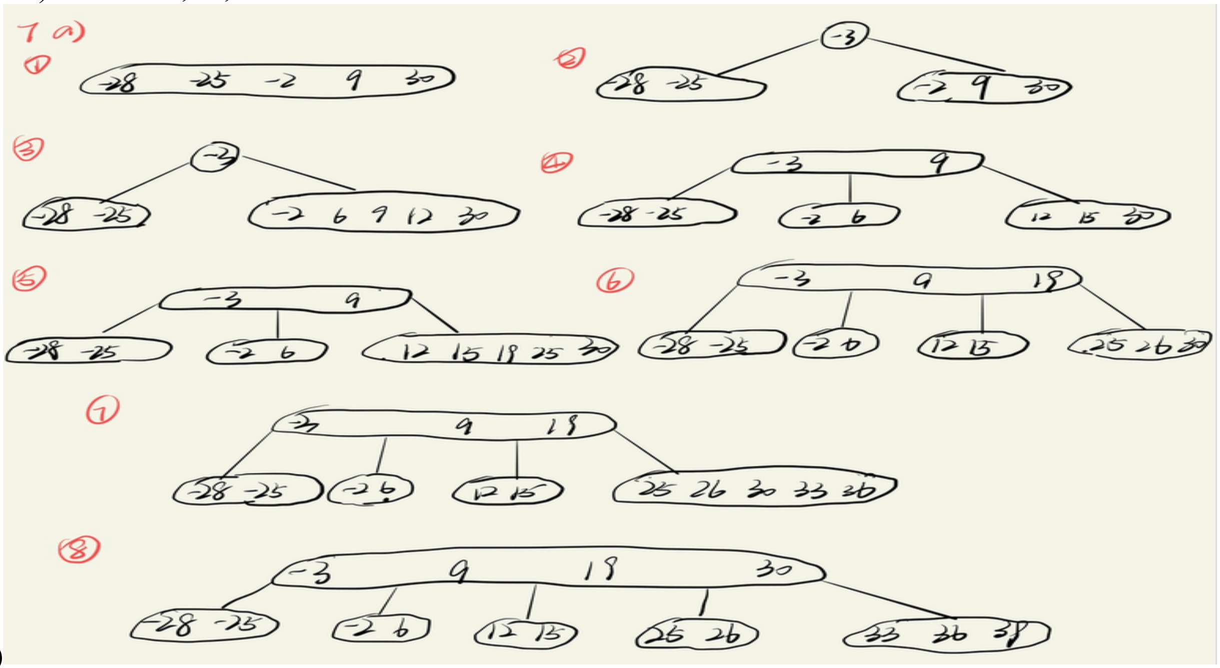
   a) Build a btree with minimum degree of 3

   b) What is the order of this btree?

   c) Why is this btree with minimum degree of 3?

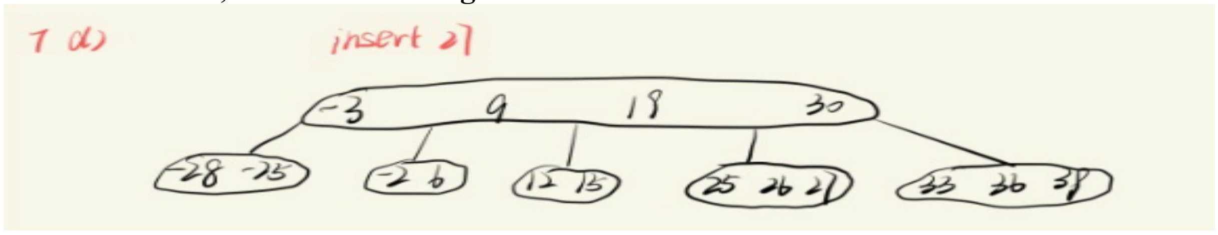
   d) Insert 27 into btree

   e) Delete 19, -2, 36

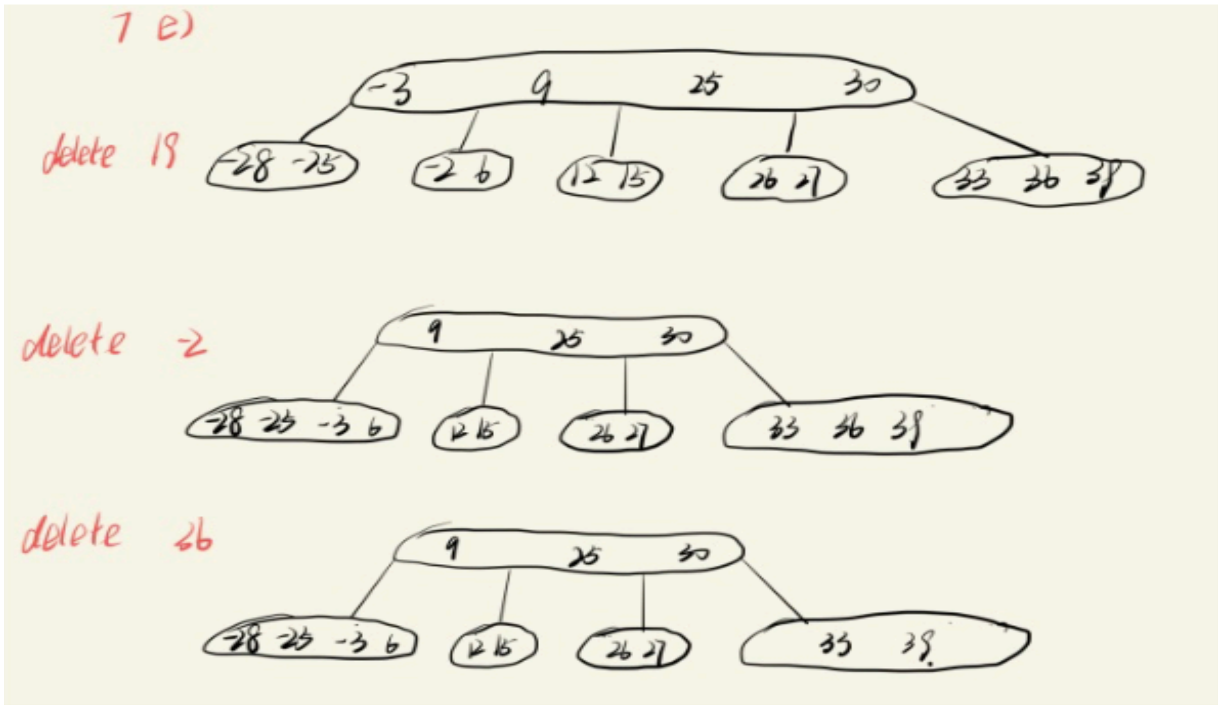


a)

b) The order of this b tree is 6
c) The minimum degree means at least children of non-leaf node.
The order means at most children of non-leaf node.
When the minimum degree is t, the order is 2t.
So when order is 6, the minimum degree is 3.



d)



e)

8. 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 2,048 bytes. Suppose that the average seek time is 5 msec, the average rotational delay is 5 msec, and the transfer rate is 100 MB per second. Suppose that a file containing 1,000,000 records of 100 bytes each is to be stored on such a disk and that no record is allowed to span two blocks.

a) How many records fit onto a block?

2048/100 ~ 20. At most 20 records in a block.

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

1,000,000/20=50,000 blocks are required to store the entire file.

9. 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,

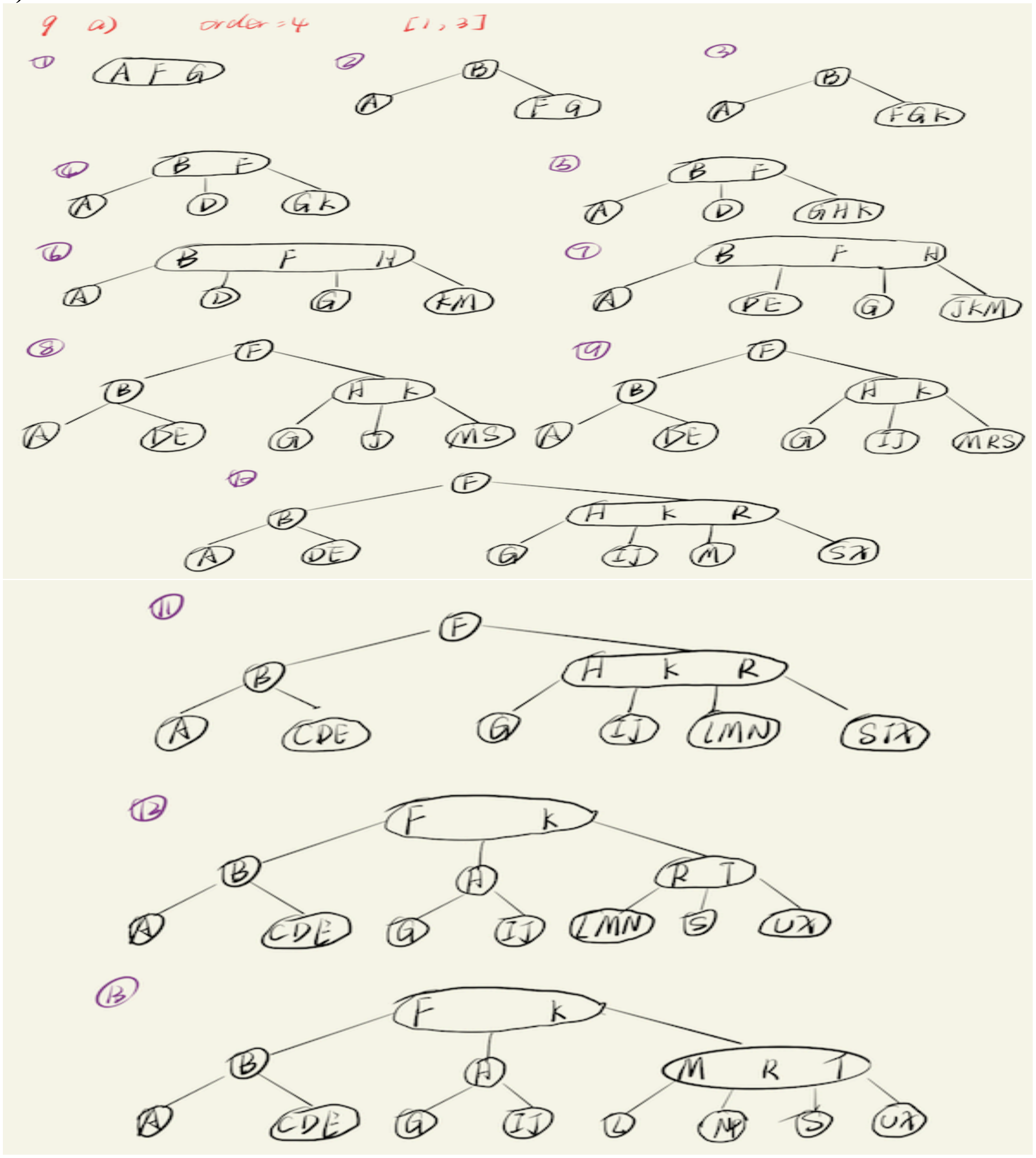
Select delete element for each of 3-cases to delete from BTree in (a)

e) Write Java code for all 3 deletion cases

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

H) Discuss height, time and space complexity

a)


9 a) order = 4 [1, 3]
① A F G
② B A F G
③ B A F G K
④ B F A D G K
⑤ B F A D G H K
⑥ B F H A D G K M
⑦ B F H A D E G J K M
⑧ F B H K A D E G J M S
⑨ F B H K A D E G I J M R S
⑩ F B H K R A D E G I J M S X
⑪ F B H K R A C D E G I J L M N S T X
⑫ F K B H R T A C D E G I J L M N S U X
⑬ F K B H M R T A C D E G I J L N P S U X

b) Minimum degree for this BTree is 2

c) The code is in H7_9c

d) Delete element for case-1: N

Delete element for case-2: R
Delete element for case-3: F
e) The code is in H7_9e
H)
Height:2

Algorithm	**Average**	**Worst case**
Space	**O(n)**	**O(n)**
Search	**O(log n)**	**O(log n)**
Insert	**O(log n)**	**O(log n)**
Delete	**O(log n)**	**O(log n)**

Appendix-A 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.

Appendix-B 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
```