Final Review (Only covers Hashing, Tree, Graph)

Please refer back to the midterm review for other topics covered in this course

Notes by Yan Yan

Final Exam Info

- Exam: Dec. 6, 2:30pm, AC 122A
- Closed book, written exam, 120 minutes. No calculators.
- Coverage: Weeks 0-12 (20-30% for Weeks 0-5)
- Types of questions: Multiple choices, True/false, short answer, long questions.
  - More details on practice and sample questions
  - Programming questions in MCQ only
- How to prepare
  - Lecture content, Assignments, Labs
  - Practice questions
  - Review and sample questions

Final Exam Rules

- Students may not leave for the first 60 minutes of the exam.
- After the first 60 minutes, students may not be allowed to enter the exam room and begin the exam.
- Students may not leave the exam room in the last 15 minutes of the exam.
- Students who need to use the washroom must be supervised.
- ALL electronics are **disallowed** including smart watches and ear buds. They should be put into your bag.
- Bags should be put under your chair.

Final Exam Rules

- Things to bring: **University of Guelph ID, pencil, eraser**.
- We only accept UoG ID for verification so you MUST bring the card with you.

- If a student is ready to submit the exam before the exam ends, they raise their hand, and an invigilator will walk to the student and get the exam.
- After the exam ends, all students shall remain in the seats, no one can leave the room until all exam papers are collected.
- Students are **not** allowed to stand up and bring their exams to the front.

Hash Table

Hash Tables

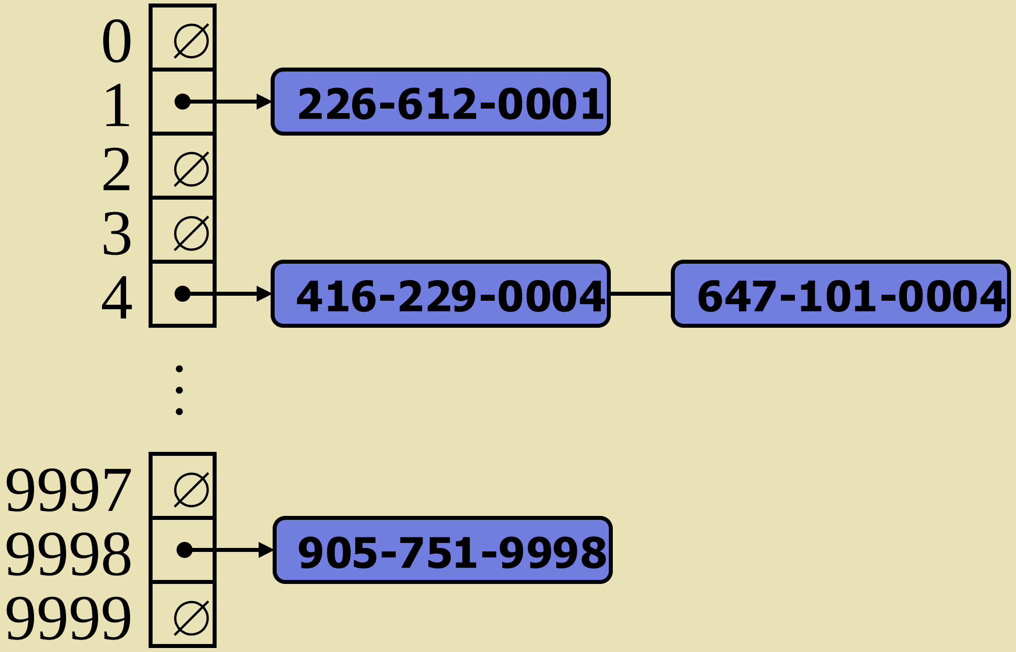
- A **hash table** is a data structure that stores unordered items by mapping (or hashing) each item to a location in an array (or vector).
- An item's **key** is the value used to map to an index.
- Each hash table array element is called a **bucket**.
  A **hash function** computes a bucket index from the item's key.

Collisions

- A **collision** occurs when an item being inserted into a hash table maps to the same bucket as an existing item in the hash table.
- **Chaining** is a collision resolution technique where each bucket has a list of items.

Example

- We design a hash table for a dictionary storing items (Phone#, Name), where a Phone# is a ten-digit positive integer
- Our hash table uses an array of size $N = 10{,}000$ and the hash function $h(x)$ = last four digits of x
- We use chaining to handle collisions



Hash Functions

- A **hash function** $\boldsymbol{h}(\,)$ maps keys of a given type to integers in a fixed interval $[0, N - 1]$
- Example:

  $$\boldsymbol{h}(x) = x \bmod N$$

  is a hash function for integer keys
- The integer returned by $\boldsymbol{h}(x)$ is called the **hash value** of key $\boldsymbol{x}$
- The goal of a hash function is to uniformly disperse keys in the range $[0, N - 1]$

Linear Probing for handling collision

- Linear probing handles collisions by placing the colliding item in the next (circularly) available table cell
- Each table cell inspected is referred to as a “probe”
- Colliding items lump together. Future collisions may cause a longer sequence of probes

Quadratic probing

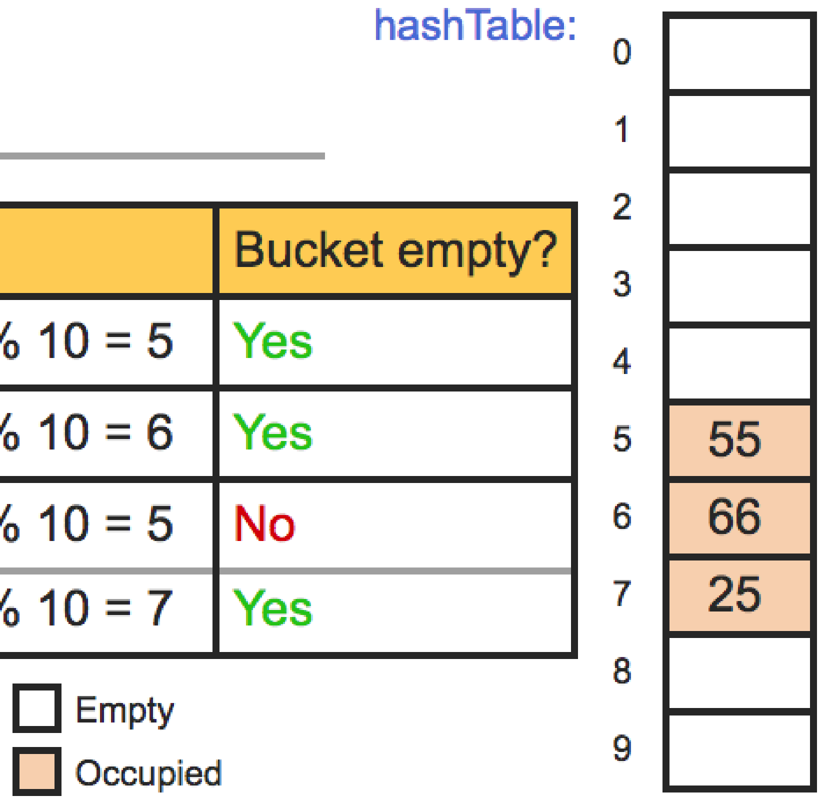
- To avoid collision, **quadratic probing (QP)** starts at the key's mapped bucket, and then quadratically searches subsequent buckets until an empty bucket is found.

$$h(x) = (H + c_1 i + c_2 i^2) \bmod (\text{table size})$$

Hash table insertion using **QP**: $c_1 = 1$ & $c_2 = 1$.

Hash function: key % 10
Quadratic probing sequence: (H + i + i * i) % 10

Operation	H(key)	i	Bucket index	Bucket empty?
Insert key 55	55 % 10 = 5	0	(5 + 0 + 0 * 0) % 10 = 5	Yes
Insert key 66	66 % 10 = 6	0	(6 + 0 + 0 * 0) % 10 = 6	Yes
Insert key 25	25 % 10 = 5	0	(5 + 0 + 0 * 0) % 10 = 5	No
		1	(5 + 1 + 1 * 1) % 10 = 7	Yes



(textbook 6.4.1)

Sample Questions:

Hash table with linear probing: Insert.

Given hash function of key % 5, determine the insert location for each item.

1) HashInsert(numsTable, item 13)

numsTable:	0	
	1	71
	2	22
	3	
	4	

Bucket =

2) HashInsert(numsTable, item 41)

numsTable:	0	
	1	21
	2	
	3	
	4	

Bucket =

3) HashInsert(numsTable, item 74)

numsTable:	0	20
	1	
	2	32
	3	
	4	84

Bucket =

Sample Questions

- Assume a hash function returns key % 16 and quadratic probing is used with $c_1 = 1$ & $c_2 = 1$. $h(x) = (H + c_1 i + c_2 i^2) \bmod (\text{table size})$
- Refer to the table below.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
32	49	16	3		99	64	23			42	11				

1) 32 was inserted before 16? True or False?

2) Which value was inserted without collision?

3) What is the probing sequence when inserting 48 into the table?

Trees

Binary Tree

- In a **binary** tree, each node has **up to** two children, known as a **left** child and a **right** child.
  - One node has up to two successors

- "Binary" means two, referring to the two children

Complete Binary Tree

- Two properties
  - every level, except possibly the last, is completely filled.
  - all nodes appear as far left as possible
- Example

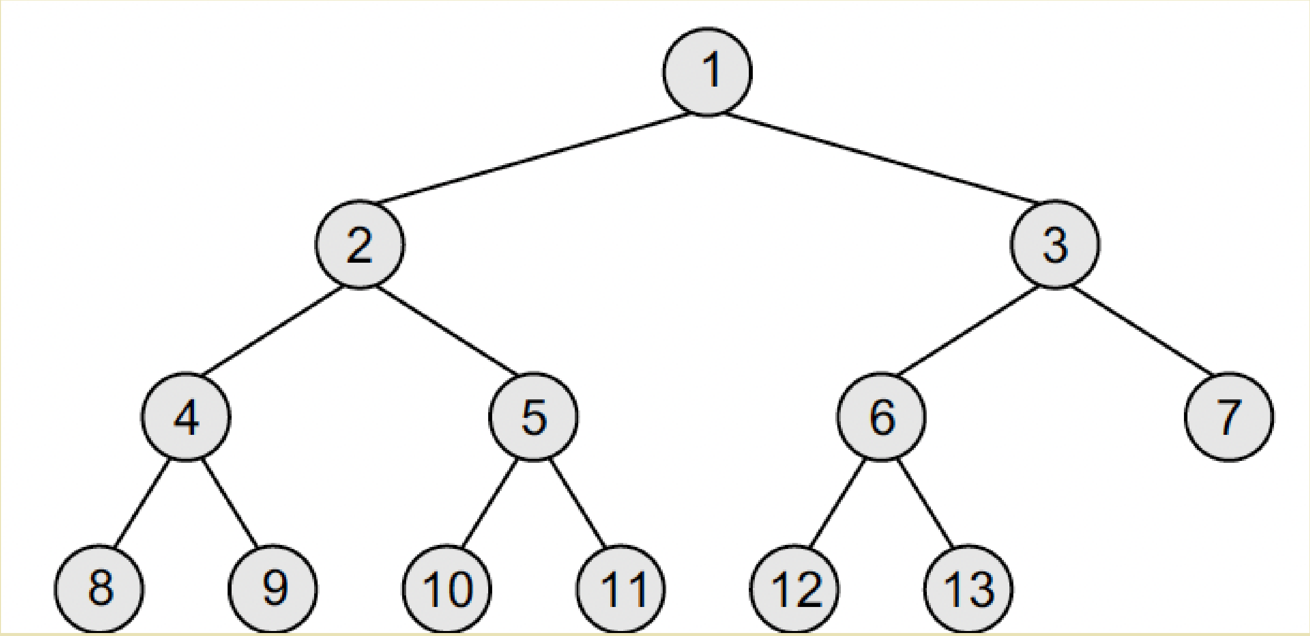


Figure from book Data Structures Using C, 2nd edition (Chapter 9, Figure 9.7), by Reema Thareja

Arithmetic Expression Tree

- Example: arithmetic expression tree for the expression $(a - b) + (c * d)$

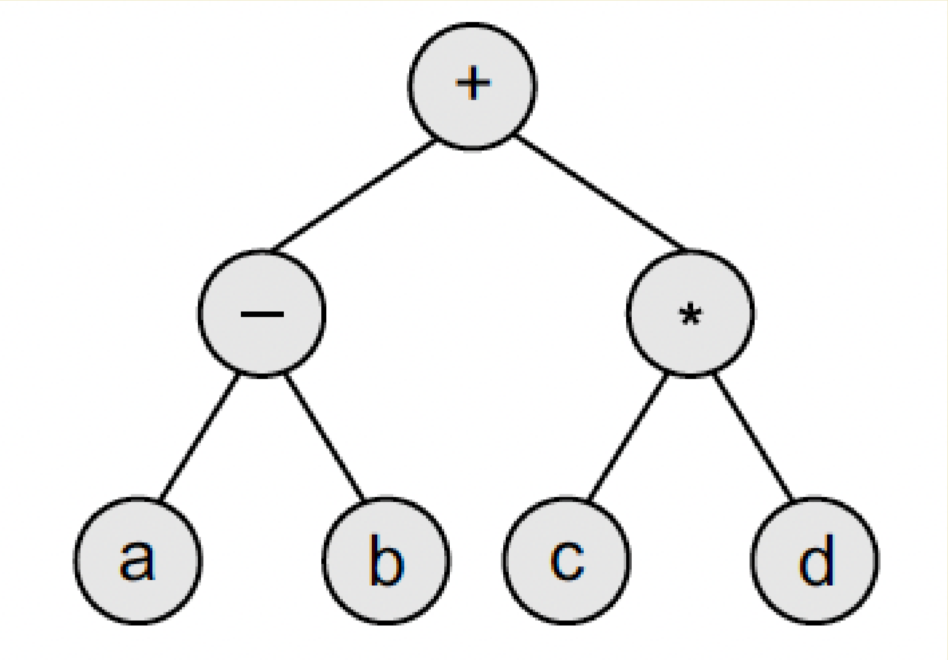


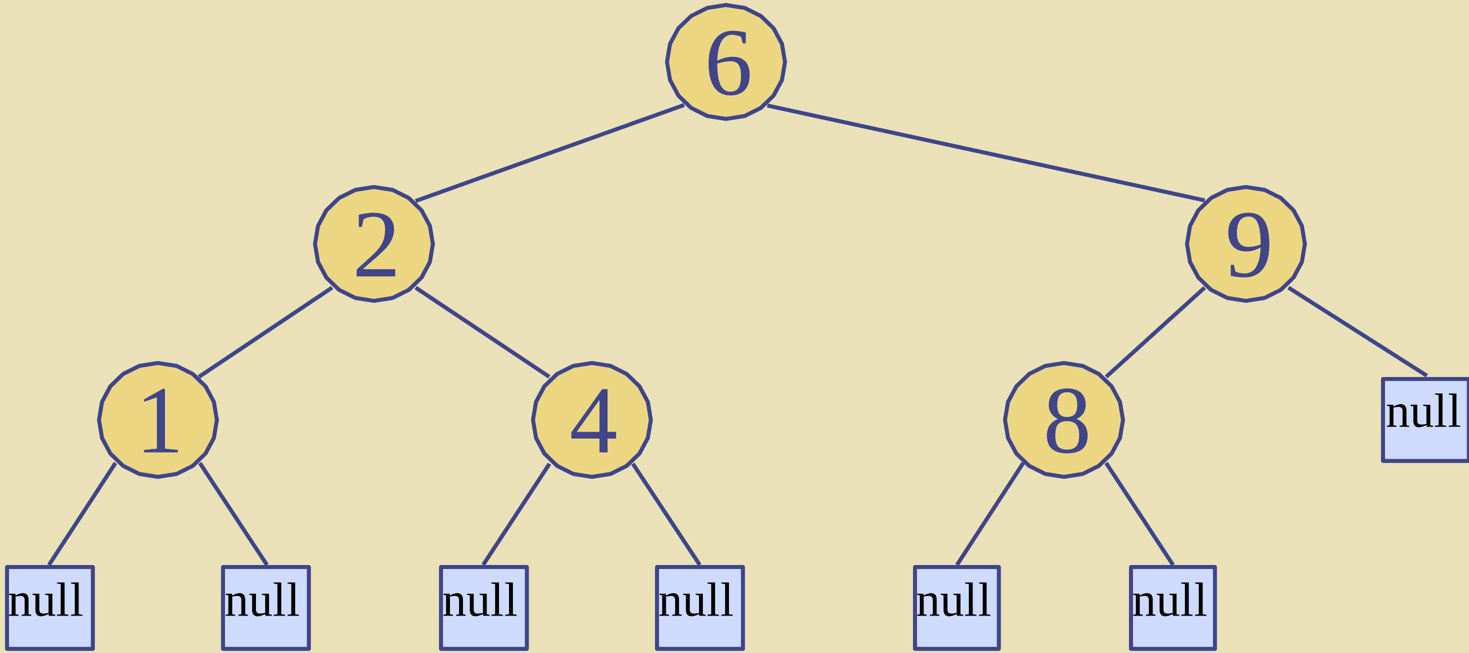
Figure from book Data Structures Using C, 2^{nd} edition (Chapter 9, Figure 9.13), by Reema Thareja

Binary Tree Traversal

- Please refer back to the slides and understand different traversals.
- Pre-order
- In-order
- Post-order
- Level-order

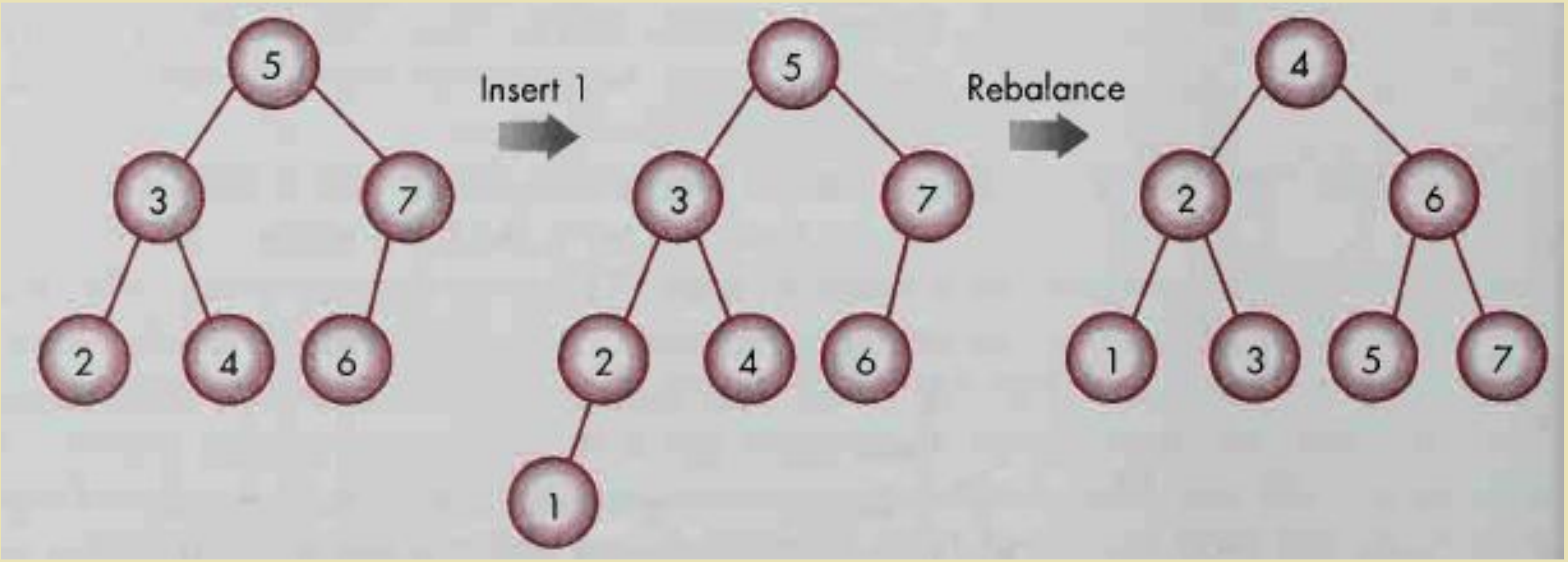
Binary Search Tree

- Nodes are arranged in an order.
- Assume no duplicates values allowed, then
  - Nodes in the left sub-tree have a value less than (<) the value of the root node.
  - Nodes in the right sub-tree have a value greater than (>) the value of root node.
- How to perform insertion and removal (refer slides)



AVL Tree

- We want to avoid the worst case in the binary search tree
- How? -- balance the tree
- Issue -- rebalance the tree may take up O(n) operations



The above figure is the Figure 9.41 from the book “Thomas A. Standish: Data Structures, Algorithms & Software Principles in C, Addison Wesley”

AVL Tree Definition

- Height – longest path from the root to some leaf
  - An empty tree has height 0 (note: some other definitions make this as -1)
  - A tree with a single node has height 1 (note: some other definitions make this as 0)

- AVL property: If N is a node in a binary tree T, we say that node N has the **AVL property** if the heights of the left and right subtrees of node N are either equal or if they differ by 1.

- Balance factor of node N, BF(N) = Height_(Nleft) - Height_(Nright)

- AVL Tree: A binary tree that each of its nodes has AVL property

- Restore AVL tree with a new node inserted (refer to slides)

Heap Definition

- A **max-heap** is a complete binary tree that every node satisfies the heap property:
  - If B is a child of A, then key(A) $\geq$ key(B)
  - (A node's key is greater than or equal to the node's children's keys).
  - In a max-heap, the root node has the highest key value in the heap
- A **min-heap** is a heap that elements at every node will be either less than or equal to the element at its left and right child.
  - the root has the lowest key value

Heap Applications

- A computer may execute jobs one at a time; upon finishing a job, the computer executes the pending job having maximum priority.
- Implement priority queue ADT
  - A priority queue is a queue where each item has a priority, and items with higher priority are closer to the front of the queue than items with lower priority.
  - **Enqueue:** insert and maintain the priority property
  - **Dequeue:** remove and return the front item of the queue (highest priority).

Heap Insertion

- Inserting the node in the tree's last level, and then swapping the node with its parent until no max-heap property violation occurs.
- Inserts fill a level (left-to-right) before adding another level, so the tree is still a complete binary tree.

Heap Removal

- A remove from a max-heap is always a removal of the root. – why?
- Replace the root with the last level's last node (so the tree remains a complete binary tree)
- Change the new root node with its greatest child until no max-heap property violation occurs.
  - Move the new node to the “correct” spot

Heap using Array

- Heaps are typically stored using arrays.
- Given a tree representation of a heap, the heap's array form is produced by traversing the tree's levels from left to right and top to bottom.
- The root **node** is always the entry at index **0** in the array, the root's left child is the entry at index 1, the root's right child is the entry at index 2, and so on.
- Downheap and upheap process (percolate-down and percolate-up) in the array implementation.

Heap using Array

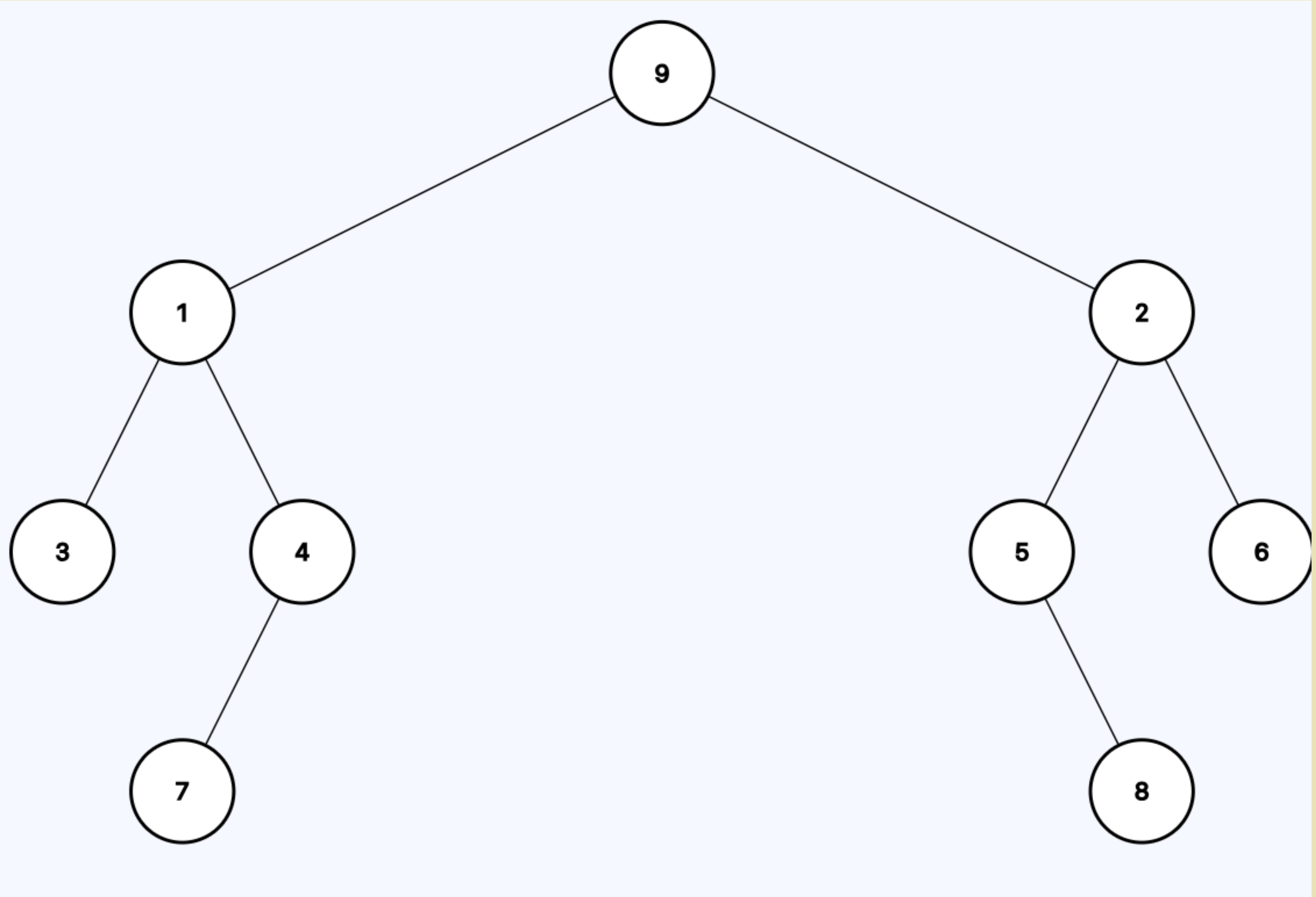
- Parent child node indices for a heap.

Node index	Parent index	Child indices
0	N/A	1, 2
1	0	3, 4
2	0	5, 6
3	1	7, 8
4	1	9, 10
5	2	11, 12
...	...	...
i	$\lfloor (i-1)/2 \rfloor$	2 * i + 1, 2 * i + 2

Figure from Zybook 9.2 Heaps

Sample question

- Given this tree, please write down the order of nodes visited using the following traversals.
- Pre-order
- In-order
- Post-order
- Level-order



Sample question

- In the following tree, what are the orders of nodes visited, if you search for
  - A. 145
  - B. 900

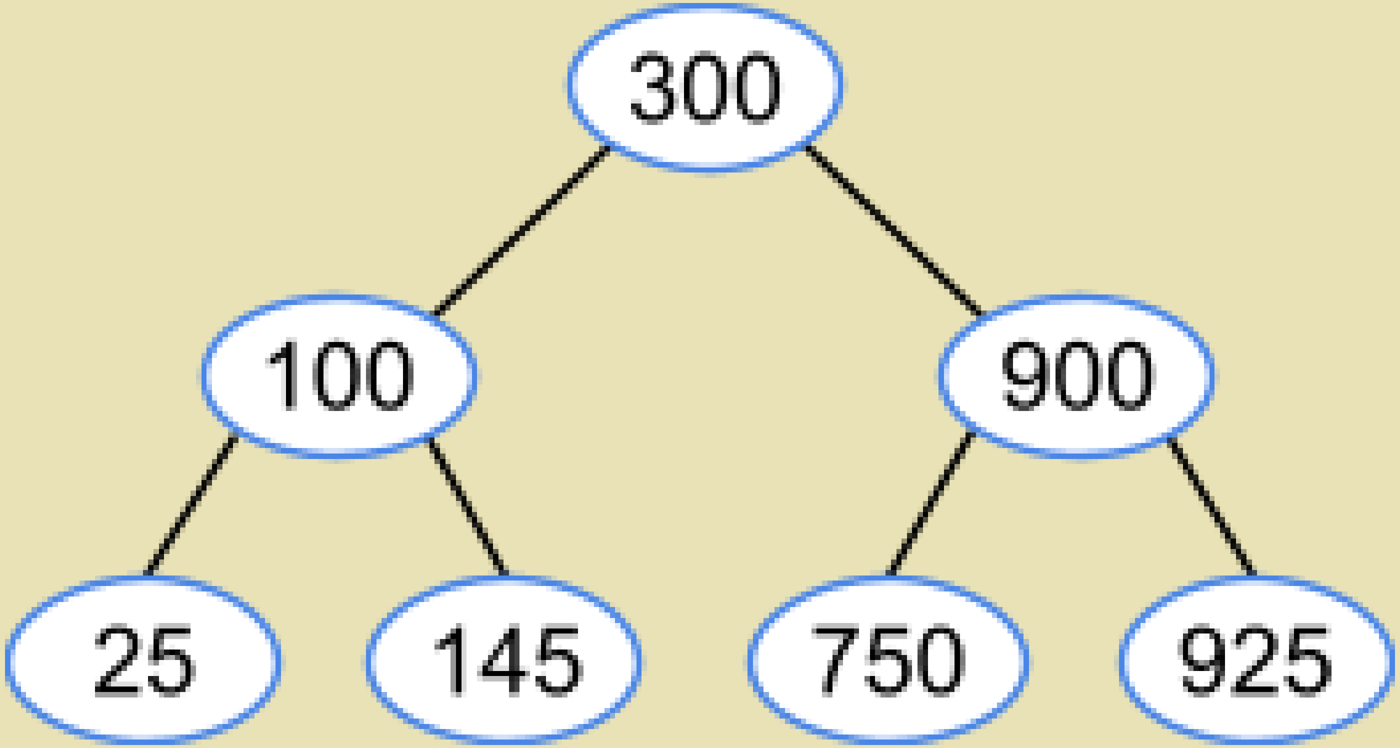
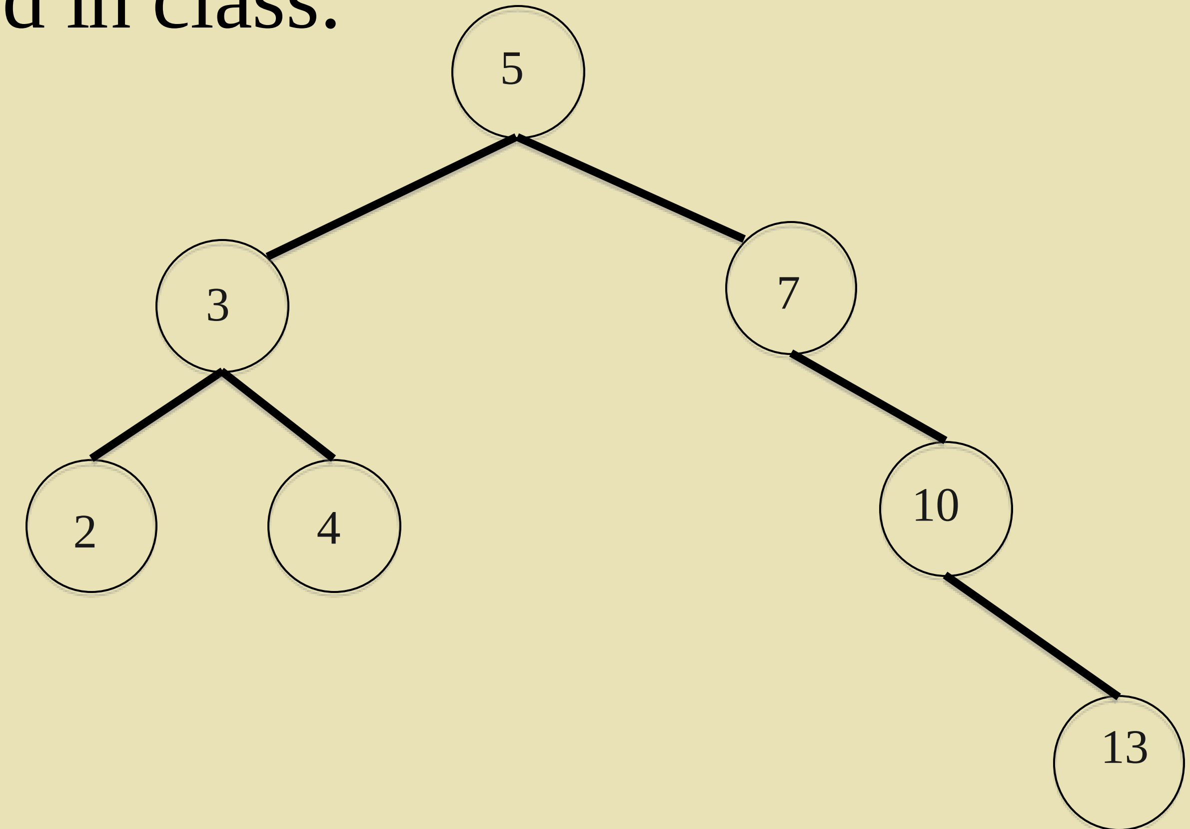


Figure from 7.3 7.3 Binary search trees

Sample question

- Which node lost its AVL tree property in this tree, and why? Draw the rebalanced AVL tree using the rotation techniques introduced in class.



Sample question

- Write down one similarity and one difference between AVL tree and Heap.
- For an AVL tree with n nodes, time complexity of insert operations is _________.

Sample question

- Remove 54 in the given heap, and using the downheap to restore the heap. Draw the heap after removing 54.
- Write down the array representation of the heap you drew.

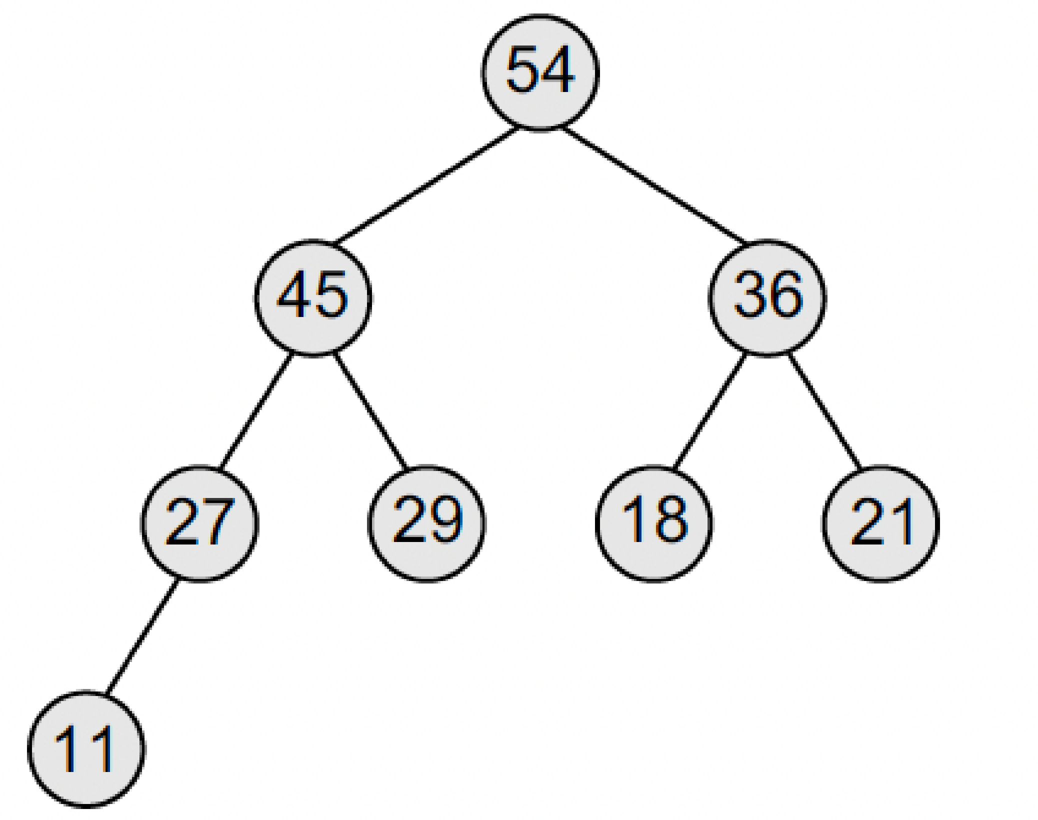


Figure from book Data Structures Using C, 2nd edition (Chapter 12, Figures 12.8), by Reema Thareja

Graphs

Key Knowledge Points and Questions

1. Given a graph, write down the adjacency list and adjacency matrix of it
2. Given a graph, write down the order of vertices visited using breadth-first search and depth-first search
3. Given a graph, perform Dijkstra’s Algorithm and Bellman-Ford's Algorithm to find the shortest path
    1. Determine when the algorithms can be applied
4. Given a graph, determine if an order is a topological ordering, and write down a valid topological ordering.
    1. Determine what graph has topological ordering

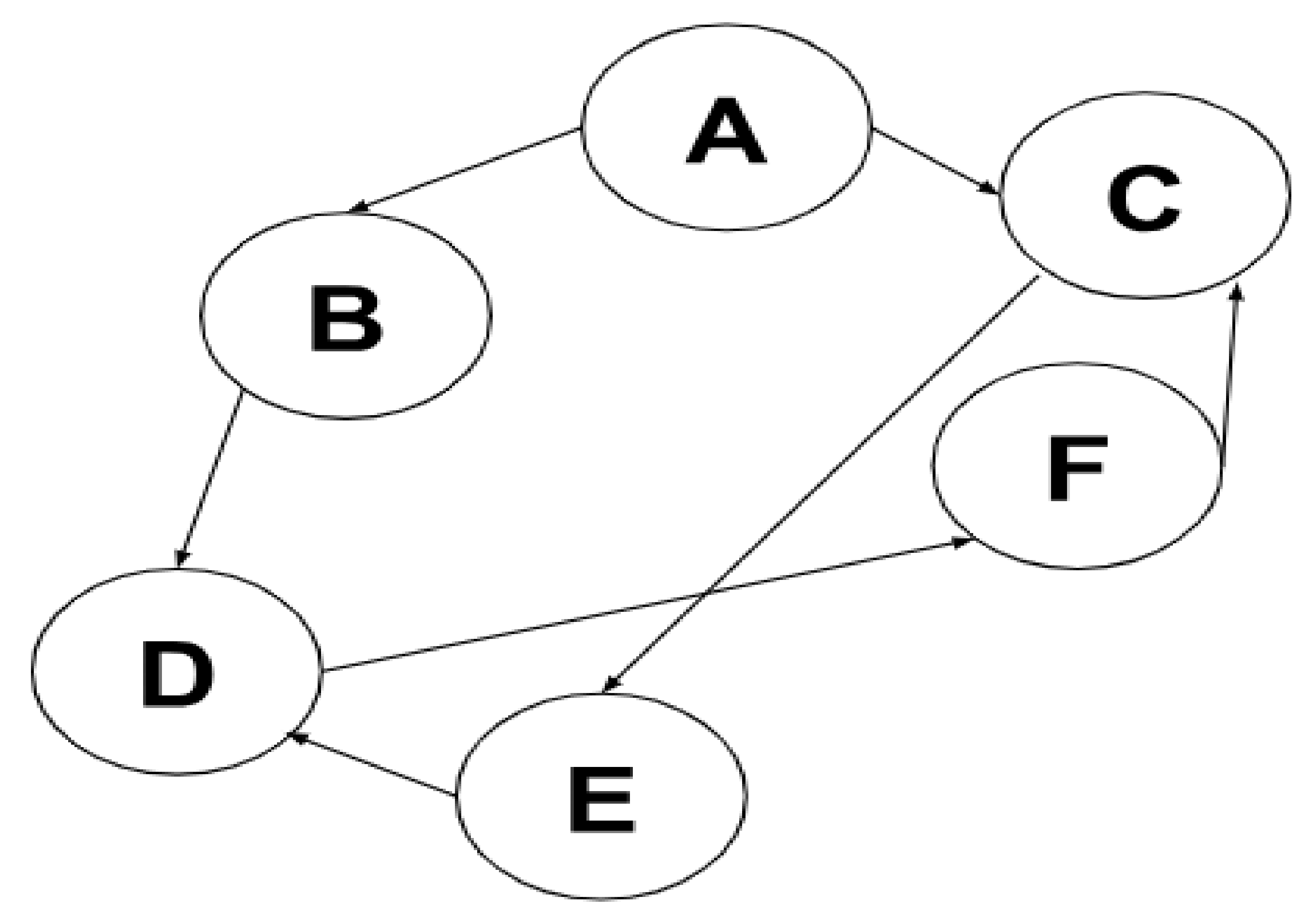
- Refer to slides to see more sample questions.

Sample question

- List two differences between Bellman-Ford's and Dijkstra's algorithm in how they can be used to find the shortest path in a graph.

Sample question

- Given the following directed graph with vertices A,B,C,D,E,F:
- Perform a Depth-First Search (DFS) starting from vertex A. List the order in which vertices are visited.
- Perform a Depth-First Search (DFS) starting from vertex A. List the order in which vertices are visited.



Final Reminder

- Please review ALL questions/examples/exercise in the lecture slides, labs, and assignments.
- Sample MCQ and T/F questions are in the practice questions.

That’s all. Good luck in your exam!