16: Binary Trees, Binary Search, Heaps, and Priority Queues
Lecture 16: Binary Trees, Binary Search, Heaps, and Priority Queues

- Binary Trees
- Tree traversals
- Binary Search
- Binary Heaps
- Priority Queues
Trees in Computer Science

- Abstract data types that store elements hierarchically rather than linearly.
- Examples of hierarchical structures:
  - Organization charts for
    - Companies (CEO at the top followed by CFO, CMO, COO, CTO, etc).
    - Universities (Board of Trustees at the top, followed by President, then by VPs, etc).
  - Sitemaps (home page links to About, Products, etc. They link to other pages).
  - Computer file systems (user at top followed by Documents, Downloads, Music, etc. Each folder can hold more folders.).
Trees in Computer Science

- Hierarchical: Each element in a tree has a **single** parent (immediate ancestor) and zero or more **children** (immediate descendants).
Definition of a tree

- A tree $T$ is a set of nodes that store elements based on a parent-child relationship:
  - If $T$ is non-empty, it has a node called the root of $T$, that has no parent.
    - Here, the root is A.
  - Each node $v$, other than the root, has a unique parent node $u$. Every node with parent $u$ is a child of $u$.
    - Here, E’s parent is C and F has two children, H and I.
Tree Terminology

- **Edge**: a pair of nodes s.t. one is the parent of the other, e.g., (K,C).
- **Parent** node is directly above **child** node, e.g., K is parent of C and N.
- **Sibling** nodes have same parent, e.g., A and F.
- K is **ancestor** of B.
- B is **descendant** of K.
- Node plus all **descendants** gives subtree.
- Nodes without descendants are called **leaves** or **external**. The rest are called **internal**.
- A set of trees is called a **forest**.
BINARY TREES

More Terminology

- **Simple path**: a series of distinct nodes s.t. there are edges between successive nodes, e.g., K-N-V-U.
- **Path length**: number of edges in path, e.g., path K-C-A has length 2.
- **Height of node**: length of longest path from the node to a leaf, e.g., N’s height is 2 (for path N-V-U).
- **Height of tree**: length of longest path from the root to a leaf. Here 3.
- **Degree of node**: number of its children, e.g., F’s degree is 2.
- **Degree of tree (arity)**: max degree of any of its nodes. Here is 2.
- **Binary tree**: a tree with arity of 2, that is any node will have 0-2 children.
**Even More Terminology**

- **Level/depth of node** defined recursively:
  - Root is at level 0.
  - Level of any other node is equal to level of parent + 1.
  - It is also known as the length of path from root or number of ancestors excluding itself.

- **Height of node** defined recursively:
  - If leaf, height is 0.
  - Else, height is max height of child + 1.
But wait there’s more!

- **Full (or proper)**: a binary tree whose every node has 0 or 2 children.

- **Complete**: a binary tree with minimal height. Any holes in tree would appear at last level to right, i.e., all nodes of last level are as left as possible.
Neither complete nor full

Complete but not full

Full but not complete

Complete and full

http://code.cloudkaksha.org/binary-tree/types-binary-tree
Practice Time: This tree is

- A: Full
- B: Complete
- C: Full and Complete
- D: Neither Full nor Complete
Answer

- A: Full
- B: Complete
- C: Full and Complete
- D: Neither Full nor Complete
Practice Time: This tree is

- A: Full
- B: Complete
- C: Full and Complete
- D: Neither Full nor Complete
Answer

- A: Full
- B: Complete
- C: Full and Complete
- D: Neither Full nor Complete
Counting in binary trees

- **Lemma**: if $T$ is a binary tree, then at level $k$, $T$ has $\leq 2^k$ nodes.

  - E.g., at level 2, at most 4 nodes (A, F, M, V)

- **Theorem**: If $T$ has height $h$, then # of nodes $n$ in $T$ satisfy:
  \[ h + 1 \leq n \leq 2^{h+1} - 1. \]

  - Equivalently, if $T$ has $n$ nodes, then $\log(n + 1) - 1 \leq h \leq n - 1$.

- **Worst case**: When $h = n - 1$ or $O(n)$, the tree looks like a left or right-leaning “stick”.

- **Best case**: When a tree is as compact as possible (e.g., complete) it has $O(\log n)$ height.
Practice Time - Problem 1 Worksheet #16

- Follow the instructions in the worksheet about the following tree:
ANSWER 1 - Worksheet #16

- **Root:** 2
- **Leaves:** 2 (in black), 10, 5, 11, 4
- **Internal nodes:** 7, 5, 6, 9
- **Siblings of 10:** 2, 6
- **Parent of 6:** 7
- **Children of 2 (in red):** 7, 5
- **Ancestors of 10:** 7 and 2 (in red)
- **Descendants of 7:** 2, 10, 6, 5, 11
- **Length of path 2-5-9-4:** 3
- **Height of 7:** 2
- **Height of tree:** 3
- **Degree of 7:** 3
- **Arity/Degree of tree:** 3
- **Level/depth of 11:** 3
Basic idea behind a simple implementation

```java
public class BinaryTree<E> {
    private Node root;

    /**
     * A node subclass which contains various recursive methods
     *
     * @param <E> The type of the contents of nodes
     */
    private class Node {
        private E element;

        private Node left;
        private Node right;

        /**
         * Node constructor with subtrees
         *
         * @param left the left node child
         * @param right the right node child
         * @param E the element contained in the node
         */
        public Node(Node left, Node right, E element) {
            this.left = left;
            this.right = right;
            this.element = item;
        }
    }
}
```
Lecture 16: Binary Trees, Binary Search, Heaps, and Priority Queues

- Binary Trees
- Tree traversals
- Binary Search
- Binary Heaps
- Priority Queues
Pre-order traversal

- Preorder(Tree)
  - Mark root as visited
  - Preorder(Left Subtree)
  - Preorder(Right Subtree)
- K C A B F D H N M V U
In-order traversal

- Inorder(Tree)
  - Inorder(Left Subtree)
  - Mark root as visited
  - Inorder(Right Subtree)
- A B C D F H K M N U V
Post-order traversal

- Postorder(Tree)
  - Postorder(Left Subtree)
  - Postorder(Right Subtree)
- Mark root as visited

B A D H F C M U V N K
Level-order traversal

- From left to right, mark nodes of level $i$ as visited before nodes in level $i + 1$. Start at level 0.

- K C N A F M V B D H U
Practice Time - Problem 2 Worksheet #16

- List the nodes in pre-order, in-order, post-order, and level order:
ANSWER Problem 2 Worksheet #16

- Pre-order: 8, 5, 9, 7, 1, 12, 2, 4, 11, 3
- In-order: 9, 5, 1, 7, 2, 12, 8, 4, 3, 11
- Post-order: 9, 1, 2, 12, 7, 5, 3, 11, 4, 8
- Level-order: 8, 5, 4, 9, 7, 11, 1, 12, 3, 2
Lecture 16: Binary Trees, Binary Search, Heaps, and Priority Queues

- Binary Trees
- Tree traversals
- Binary Search
- Binary Heaps
- Priority Queues
Binary search

- **Goal**: Given a sorted array and a key, find index of the key in the array.

- **Basic mechanism**: Compare key against middle entry.
  - If too small, repeat in left half.
  - If too large, repeat in right half.
  - If equal, you are done.
Binary search implementation

- First binary search published in 1946.
- First bug-free one in 1962.

```java
public static int binarySearch(int[] a, int key) {
    int lo = 0, hi = a.length-1;
    while (lo <= hi) {
        int mid = lo + (hi - lo) / 2;
        if (key < a[mid])
            hi = mid - 1;
        else if (key > a[mid])
            lo = mid + 1;
        else return mid;
    }
    return -1;
}
```
- Uses at most $1 + \log n$ key compares to search in a sorted array of size $n$, that is it is $O(\log n)$.  


Lecture 16: Binary Trees, Binary Search, Heaps, and Priority Queues

- Binary Trees
- Tree traversals
- Binary Search
- Binary Heaps
- Priority Queues
Heap-ordered binary trees

- A binary tree is **heap-ordered** if the key in each node is larger than or equal to the keys in that node’s two children (if any).

- Equivalently, the key in each node of a heap-ordered binary tree is smaller than or equal to the key in that node’s parent (if any).

- No assumption of which child is smaller.

- Moving up from any node, we get a non-decreasing sequence of keys.

- Moving down from any node we get a non-increasing sequence of keys.
Heap-ordered binary trees

- The largest key in a heap-ordered binary tree is found at the root!
Binary heap representation

- We could use a linked representation but we would need three links for every node (one for parent, one for left subtree, one for right subtree).

- If we use complete binary trees, we can use instead an array.
  - Compact arrays vs explicit links means memory savings!
Binary heaps

- **Binary heap**: the array representation of a complete heap-ordered binary tree.
  - Items are stored in an array such that each key is guaranteed to be larger (or equal to) than the keys at two other specific positions (children).
  - Max-heap but there are min-heaps, too.
Array representation of heaps

- Nothing is placed at index 0.
- Root is placed at index 1.
- Rest of nodes are placed in level order.
- No unnecessary indices and no wasted space because it’s complete.
Reuniting immediate family members.

- For every node at index $k$, its parent is at index $\lfloor k/2 \rfloor$.
- Its two children are at indices $2k$ and $2k + 1$.
- We can travel up and down the heap by using this simple arithmetic on array indices.
Swim/promote/percolate up/bottom up reheapify

- Scenario: a key becomes larger than its parent therefore it violates the heap-ordered property.

- To eliminate the violation:
  - Exchange key in child with key in parent.
  - Repeat until heap order restored.
Swim/promote/percolate up

```java
private void swim(int k) {
    while (k > 1 && a[k/2].compareTo(a[k])<0) {
        E temp = a[k];
        a[k] = a[k/2];
        a[k/2] = temp;
        k = k/2;
    }
}
```
Binary heap: insertion

- **Insert**: Add node at end in bottom level, then swim it up.

- **Cost**: At most $\log n + 1$ compares.

```java
public void insert(E x) {
    a[++n] = x;
    swim(n);
}
```
Practice Time - Problem 3 Worksheet #16

- Insert 47 in this binary heap.
ANSWER 3
Sink/demote/top down heapify

- Scenario: a key becomes smaller than one (or both) of its children’s keys.

- To eliminate the violation:
  - Exchange key in parent with key in \textbf{larger} child.
  - Repeat until heap order is restored.
Sink/demote/top down heapify

```java
private void sink(int k) {
    while (2*k <= n) {
        int j = 2*k;
        if (j < n && a[j].compareTo(a[j+1])<0))
            j++;
        if (a[k].compareTo(a[j])>=0))
            break;
        E temp = a[k];
        a[k] = a[j];
        a[j] = temp;
        k = j;
    }
}
```
Sink 7 to its appropriate place in this binary heap.
BINARY HEAP

ANSWER 4
Binary heap: return (and delete) the maximum

- **Delete max**: Exchange root with node at end. Return it and delete it. Sink the new root down.

- **Cost**: At most $2 \log n$ compares.
Binary heap: delete and return maximum

[Diagram of a binary heap process, showing the steps of removing the maximum key:]
Practice Time - Problem 5 Worksheet #16

- Delete max (and return it!)
BINARY HEAP

ANSWER 5
Things to remember about running time complexity of heaps

- Insertion is $O(\log n)$.
- Delete max is $O(\log n)$.
- Space efficiency is $O(n)$. 
2.4 Binary Heap Demo
Lecture 16: Binary Trees, Binary Search, Heaps, and Priority Queues

- Binary Trees
- Tree traversals
- Binary Search
- Binary Heaps
- Priority Queues
Priority Queue ADT

- Two operations:
  - Delete the maximum
  - Insert

- Applications: load balancing and interruption handling in OS, Huffman codes for compression, A* search for AI, Dijkstra’s and Prim's algorithm for graph search, etc.

- How can we implement a priority queue efficiently?
Option 1: Unordered array

- The *lazy* approach where we defer doing work (deleting the maximum) until necessary.
- Insert is $O(1)$ and assumes we have the space in the array.
- Delete maximum is $O(n)$ (have to traverse the entire array to find the maximum element and exchange it with the last element).
public class UnorderedArrayMaxPQ<Key extends Comparable<Key>> {
    private Key[] pq;       // elements
    private int n;          // number of elements

    // set initial size of heap to hold size elements
    public UnorderedArrayMaxPQ(int capacity) {
        pq = (Key[]) new Comparable[capacity];
        n = 0;
    }

    public boolean isEmpty() { return n == 0; }
    public int size() { return n; }
    public void insert(Key x) { pq[n++] = x; }

    public Key delMax() {
        int max = 0;
        for (int i = 1; i < n; i++){
            if (pq[max].compareTo(pq[i]) < 0) {
                max = i;
            }
        }
        Key temp = pq[max];
        pq[max] = pq[n-1];
        pq[n-1] = temp;

        return pq[--n];
    }
}
Practice Time

- Given an empty array of capacity 10, perform the following operations in a priority queue based on an unordered array (lazy approach):

  1. Insert P
  2. Insert Q
  3. Insert E
  4. Delete max
  5. Insert X
  6. Insert A
  7. Insert M
  8. Delete max
  9. Insert P
  10. Insert L
  11. Insert E
  12. Delete max
Answer

insert P
insert Q
insert E
delete-max → Q
insert X
insert A
insert M
delete-max → X
insert P
insert L
insert E
delete-max → P
Option 2: Ordered array

- The *eager* approach where we do the work (keeping the array sorted) up front to make later operations efficient.
- Insert is $O(n)$ (we have to find the index to insert and shift elements to perform insertion).
- Delete maximum is $O(1)$ (just take the last element which will be the maximum).
public class OrderedArrayMaxPQ<Key extends Comparable<Key>> {
    private Key[] pq;       // elements
    private int n;           // number of elements

    // set initial size of heap to hold size elements
    public OrderedArrayMaxPQ(int capacity) {
        pq = (Key[]) (new Comparable[capacity]);
        n = 0;
    }

    public boolean isEmpty() { return n == 0; }
    public int size() { return n; }
    public Key delMax() { return pq[--n]; }

    public void insert(Key key) {
        int i = n - 1;
        while (i >= 0 && key.compareTo(pq[i]) < 0) {
            pq[i+1] = pq[i];
            i--;
        }
        pq[i+1] = key;
        n++;
    }
}
Practice Time

- Given an empty array of capacity 10, perform the following operations in a priority queue based on an ordered array (eager approach):

  1. Insert P
  2. Insert Q
  3. Insert E
  4. Delete max
  5. Insert X
  6. Insert A
  7. Insert M
  8. Delete max
  9. Insert P
  10. Insert L
  11. Insert E
  12. Delete max
Answer

P
0 1 2 3 4 5 6 7 8 9

Q
0 1 2 3 4 5 6 7 8 9

E
0 1 2 3 4 5 6 7 8 9

P
0 1 2 3 4 5 6 7 8 9

Q
0 1 2 3 4 5 6 7 8 9

E
0 1 2 3 4 5 6 7 8 9

P
0 1 2 3 4 5 6 7 8 9

X
0 1 2 3 4 5 6 7 8 9

A
0 1 2 3 4 5 6 7 8 9

E
0 1 2 3 4 5 6 7 8 9

P
0 1 2 3 4 5 6 7 8 9

X
0 1 2 3 4 5 6 7 8 9

A
0 1 2 3 4 5 6 7 8 9

E
0 1 2 3 4 5 6 7 8 9

P
0 1 2 3 4 5 6 7 8 9

M
0 1 2 3 4 5 6 7 8 9

E
0 1 2 3 4 5 6 7 8 9

P
0 1 2 3 4 5 6 7 8 9

X
0 1 2 3 4 5 6 7 8 9

A
0 1 2 3 4 5 6 7 8 9

M
0 1 2 3 4 5 6 7 8 9

P
0 1 2 3 4 5 6 7 8 9

L
0 1 2 3 4 5 6 7 8 9

E
0 1 2 3 4 5 6 7 8 9

P
0 1 2 3 4 5 6 7 8 9

E
0 1 2 3 4 5 6 7 8 9

P
0 1 2 3 4 5 6 7 8 9

P
0 1 2 3 4 5 6 7 8 9

Answer:

1. Insert P
2. Insert Q
3. Insert E
4. Delete-max -> Q
5. Insert X
6. Insert A
7. Insert M
8. Delete-max -> X
9. Insert P
10. Insert L
11. Insert E
12. Delete-max -> P
Option 3: Binary heap

- Will allow us to both insert and delete max in $O(\log n)$ running time.
- There is no way to implement a priority queue in such a way that insert and delete max can be achieved in $O(1)$ running time.
- Priority queues are synonyms to binary heaps.
Practice Time

- Given an empty binary heap that represents a priority queue, perform the following operations:

1. Insert P
2. Insert Q
3. Insert E
4. Delete max
5. Insert X
6. Insert A
7. Insert M
8. Delete max
9. Insert P
10. Insert L
11. Insert E
12. Delete max
Answer
Lecture 16: Binary Trees, Binary Search, Heaps, and Priority Queues

- Binary Trees
- Tree traversals
- Binary Search
- Binary Heaps
- Priority Queues
Readings:

- Recommended Textbook:
  - Chapter 2.4 (Pages 308-327)

- Website:

- Visualization:
  - Insert and ExtractMax: [https://visualgo.net/en/heap](https://visualgo.net/en/heap)

Worksheet

- Lecture 16 worksheet
Practice Problem 1

Given the tree below, list the nodes in order of visit in a:

- pre-order traversal
- in-order traversal
- post-order traversal
- level-order traversal
Practice Problem 2

- Given the binary heap below, delete and return the max.
Practice Problem 3

- Suppose that the sequence 16, 18, 9, 15, *, 18, *, *, 9, *, 20, *
  25, *, *, *, 17, 21, 5, *, *, *, 21, *, 5 (where a number means insert and an asterisk means delete the maximum) is applied to an initially empty priority queue. Give the sequence of numbers returned by the delete maximum operations.
ASSIGNED READINGS AND PRACTICE PROBLEMS

ANSWER 1

- pre-order: 90, 36, 25, 2, 3, 26, 19, 17, 7, 1
- in-order: 2, 25, 3, 36, 19, 26, 90, 7, 17, 1
- post-order: 2, 3, 25, 19, 26, 36, 7, 1, 17, 90
- level-order: 90, 36, 17, 25, 26, 7, 1, 2, 3, 19
Given the binary heap below, delete and return the max.
ANSWER 3

Suppose that the sequence 16, 18, 9, 15, *, 18, *, *, 9, *, 20, *, 25, *, *, *, 17, 21, 5, *, *, *, 21, *, 5 (where a number means insert and an asterisk means delete the maximum) is applied to an initially empty priority queue. Give the sequence of numbers returned by the delete maximum operations.

- 18, 18, 16, 15, 20, 25, 9, 9, 21, 17, 5, 21