



MERRIMACK COLLEGE

CSC6023 - Advanced Algorithms

Advanced Data Structures

Patrick Neff

Advanced Data Structures



MERRIMACK COLLEGE



This is a very fast paced course, After this four week topic and we are at the half the course. So, let's see the greedy algorithms today.

Agenda Presentation

Tree Review and AVL Trees

- Types of Trees
- AVL Trees
- Code for AVL Trees
- Worksheet Assignment for AVL Trees

Graphs

- Types of Graphs
- Common Graph Applications
- Common Problems with Graphs
- Dijkstra's Algorithm
- Dijkstra's Algorithm Code
- Dijkstra's Algorithm Example
- Programming Assignment for Dijkstra's Algorithm



Tree Review:

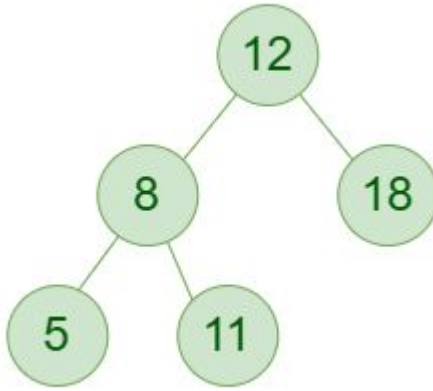
- **Full Binary Tree:** Every node has either 0 or 2 child nodes, i.e., left and right or no children
- **Complete Binary Tree:** All levels, except possibly the last, are filled, and all nodes are as left as possible
- **Perfect Binary Tree:** All nodes have exactly two children, and all leaf nodes are at the same level
- **Balanced Binary Tree:** The heights of any node's left and right subtrees differ by at most one
- **Binary Search Tree (BST):** A binary tree in which: (1) The left subtree of a node contains only nodes with keys lesser than the node's key. (2) The right subtree of a node contains only nodes with keys greater than the node's key. (3) The left and right subtree each must also be a binary search tree.

Source for definitions and images of trees on slides 26-30:

(<https://www.geeksforgeeks.org/binary-tree-data-structure/?ref=lbp>)

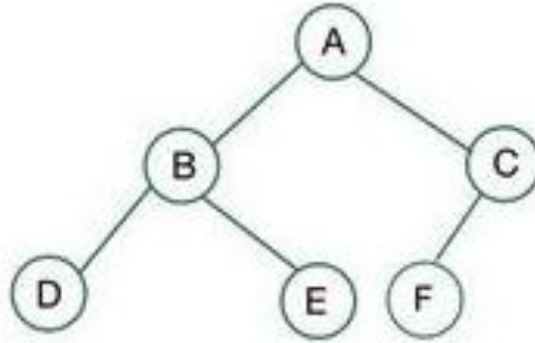
Tree Review:

- **Full Binary Tree:** Every node has either 0 or 2 child nodes, i.e., left and right or no children



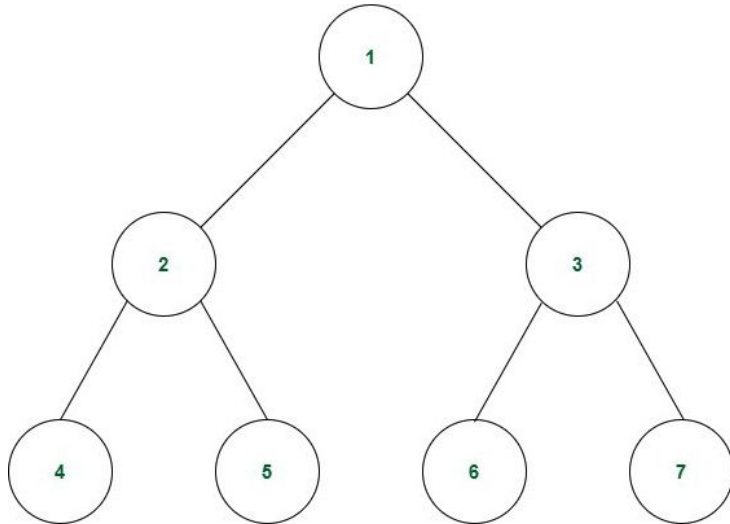
Tree Review:

- **Complete Binary Tree:** All levels, except possibly the last, are filled, and all nodes are as left as possible



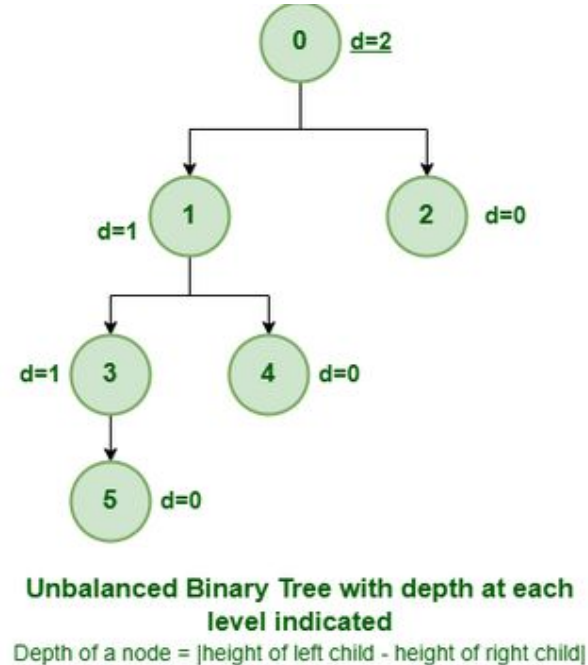
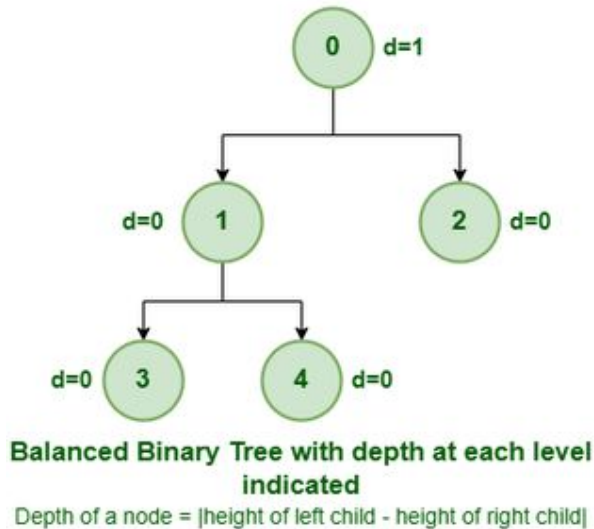
Tree Review:

- **Perfect Binary Tree:** All nodes (except leaf nodes) have exactly two children, and all leaf nodes are at the same level



Tree Review:

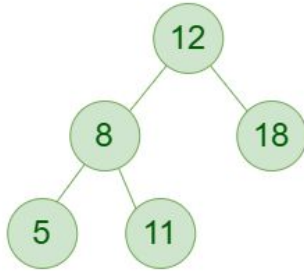
- **Balanced Binary Tree:** The heights of any node's left and right subtrees differ by at most one



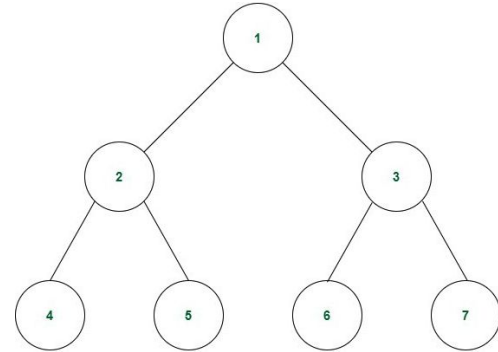
Tree Review:

- **Binary Search Tree (BST):** A binary tree in which: (1) The left subtree of a node contains only nodes with keys lesser than the node's key. (2) The right subtree of a node contains only nodes with keys greater than the node's key. (3) The left and right subtree each must also be a binary search tree.

Binary Search Tree:



Not a Binary Search Tree:



Transform-and-Conquer Algorithms

Third Example

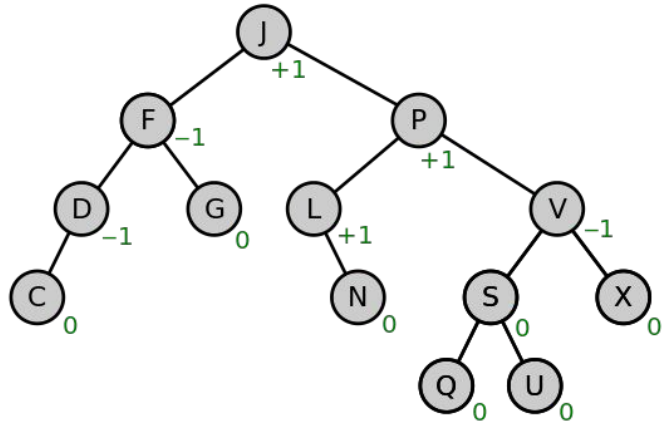
AVL Trees

- Named after the authors: Adelson, Velsky, and Landis (1962)
- Each insertion (or deletion) may require balancing - called **rotation**
 - Simple rotation
 - Left rotation or Right rotation
 - With or without children
 - Double rotation
 - Right-left rotation or Left-right rotation
 - With or without children
- A formal definition:
 - https://en.wikipedia.org/wiki/AVL_tree



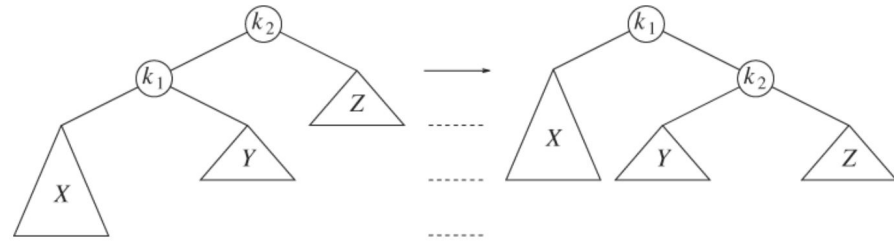
Transform-and-Conquer Algorithms

Third Example

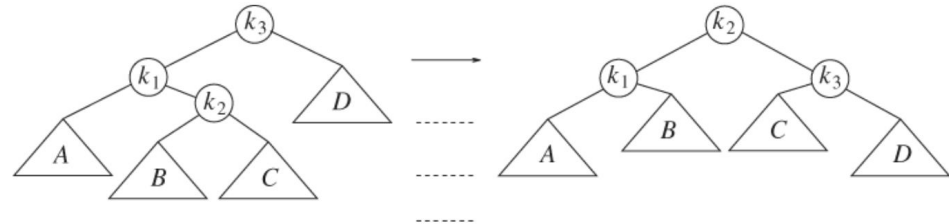


AVL Trees

- Single Rotation

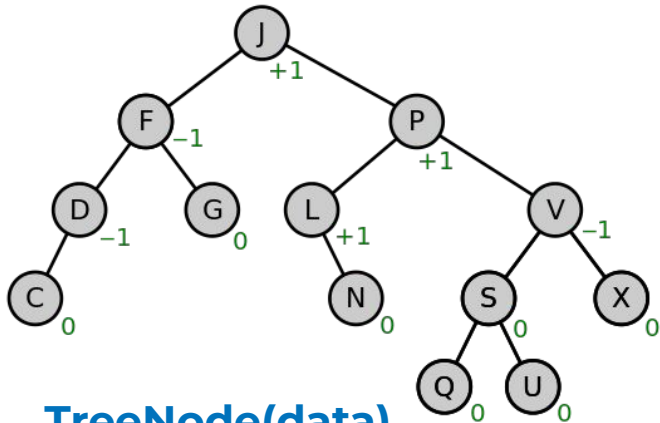


- Double Rotation



Transform-and-Conquer Algorithms

Third Example



TreeNode(data)
AVLTree()



MERRIMACK COLLEGE

AVL Trees

```
# Create a tree node
class TreeNode(object):
    def __init__(self, data):
        self.data = data
        self.left = None
        self.right = None
        self.height = 1

class AVLTree(object):
    # Function to insert a node
    def insert_node(self, root, data):
        #...
        #...
        return root

def main():
    myTree = AVLTree()
    root = None
    a = [33, 13, 52, 9, 21, 61, 8, 11]
    for d in a:
        root = myTree.insert_node(root, d)
```

- A tree is defined by a root node (and its children)
- No need to have a tree constructor
- Just inserting a node in a root `TreeNode` that may be `None` (empty) or not

- Implementation details:

<https://www.programiz.com/dsa/avl-tree>

Transform-and-Conquer Algorithms

Third Example

```
# Create a tree node
class TreeNode(object):
    def __init__(self, data):
        self.data = data
        self.left = None
        self.right = None
        self.height = 1
```

insert_node(root, data)



MERRIMACK COLLEGE

AVL Trees

- If the root is empty, insert in the root, otherwise, insert left or right
- Update the height
- If unbalanced (tallest right - balanceFactor > 1), rotate right
- If unbalanced (tallest left - balanceFactor < -1), rotate left

```
# Function to insert a node
def insert_node(self, root, data):

    # Find the correct location and insert the node
    if not root:
        return TreeNode(data)
    elif data < root.data:
        root.left = self.insert_node(root.left, data)
    else:
        root.right = self.insert_node(root.right, data)

    root.height = 1 + max(self.getHeight(root.left),
                          self.getHeight(root.right))

    # Update the balance factor and balance the tree
    balanceFactor = self.getBalance(root)
    if balanceFactor > 1:
        if data < root.left.data:
            return self.rightRotate(root)
        else:
            root.left = self.leftRotate(root.left)
            return self.rightRotate(root)

    if balanceFactor < -1:
        if data > root.right.data:
            return self.leftRotate(root)
        else:
            root.right = self.rightRotate(root.right)
            return self.leftRotate(root)

    return root
```

Transform-and-Conquer Algorithms

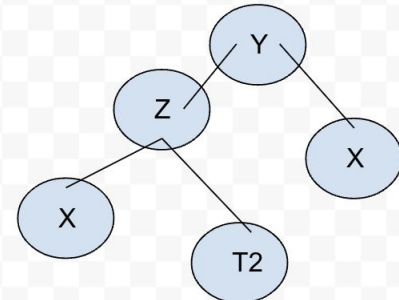
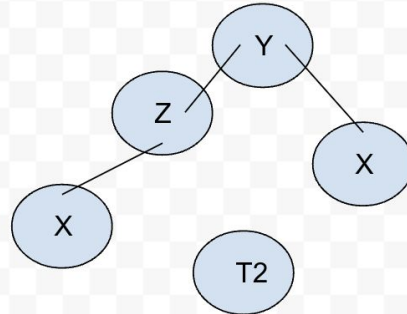
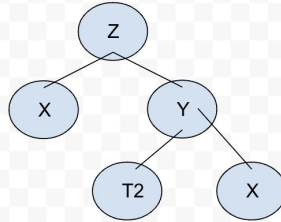
AVL Trees

Third Example

```
# Create a tree node
class TreeNode(object):
    def __init__(self, data):
        self.data = data
        self.left = None
        self.right = None
        self.height = 1
```

leftRotate(z)

```
# Function to perform left rotation
def leftRotate(self, z):
    y = z.right
    T2 = y.left
    y.left = z
    z.right = T2
    z.height = 1 + max(self.getHeight(z.left),
                       self.getHeight(z.right))
    y.height = 1 + max(self.getHeight(y.left),
                       self.getHeight(y.right))
    return y
```



Transform-and-Conquer Algorithms

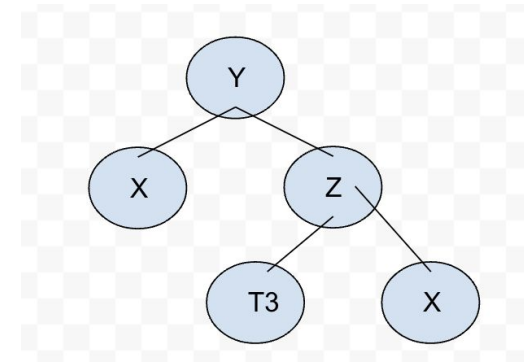
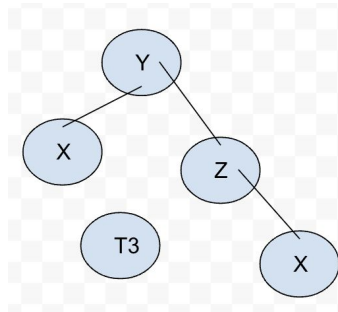
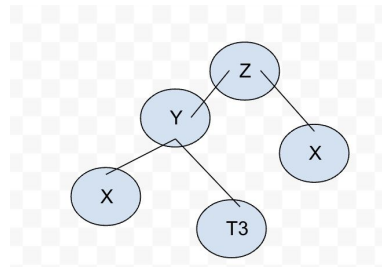
AVL Trees

Third Example

```
# Create a tree node
class TreeNode(object):
    def __init__(self, data):
        self.data = data
        self.left = None
        self.right = None
        self.height = 1
```

rightRotate(z)

```
# Function to perform right rotation
def rightRotate(self, z):
    y = z.left
    T3 = y.right
    y.right = z
    z.left = T3
    z.height = 1 + max(self.getHeight(z.left),
                        self.getHeight(z.right))
    y.height = 1 + max(self.getHeight(y.left),
                       self.getHeight(y.right))
    return y
```



Transform-and-Conquer Algorithms

Third Example

```
# Create a tree node
class TreeNode(object):
    def __init__(self, data):
        self.data = data
        self.left = None
        self.right = None
        self.height = 1
```

getHeight(root)
getBalance(root)



MERRIMACK COLLEGE

AVL Trees

```
# Get the height of the node
def getHeight(self, root):
    if not root:
        return 0
    return root.height

# Get balance factor of the node
def getBalance(self, root):
    if not root:
        return 0
    return self.getHeight(root.left) - self.getHeight(root.right)
```

- Recursive implementations used in the insertion and deletion functions
 - insert_node
 - delete_node

Let's run some code. demo2

Transform-and-Conquer Algorithms

Third Example

```
# Create a tree node
class TreeNode(object):
    def __init__(self, data):
        self.data = data
        self.left = None
        self.right = None
        self.height = 1
```

delete_node(root, data)

AVL Trees

- If empty, do nothing
- If smaller search left
- If greater, search right
- If the node is the one to delete, pull right if left is empty, or pull left if right is empty, otherwise, pull the minimum data at right

```
# Function to delete a node
def delete_node(self, root, data):

    # Find the node to be deleted and remove it
    if not root:
        return root
    elif data < root.data:
        root.left = self.delete_node(root.left, data)
    elif data > root.data:
        root.right = self.delete_node(root.right, data)
    else:
        if root.left is None:
            temp = root.right
            root = None
            return temp
        elif root.right is None:
            temp = root.left
            root = None
            return temp
        temp = self.getMinValueNode(root.right)
        root.data = temp.data
        root.right = self.delete_node(root.right,
                                      temp.data)
    if root is None:
        return root
```



Transform-and-Conquer Algorithms

Third Example

```
# Create a tree node
class TreeNode(object):
    def __init__(self, data):
        self.data = data
        self.left = None
        self.right = None
        self.height = 1
```

getMinValueNode(root)
getMaxValueNode(root)

AVL Trees

```
def getMinValueNode(self, root):
    if root is None or root.left is None:
        return root
    return self.getMinValueNode(root.left)

def getMaxValueNode(self, root):
    if root is None or root.right is None:
        return root
    return self.getMaxValueNode(root.right)
```

- Recursive implementations used in the deletion function (and also available to search)
 - delete_node



Transform-and-Conquer Algorithms

Third Example

```
# Create a tree node
class TreeNode(object):
    def __init__(self, data):
        self.data = data
        self.left = None
        self.right = None
        self.height = 1
```

printPreOrder(root)
printInOrder(root)



MERRIMACK COLLEGE

AVL Trees

```
def printPreOrder(self, root):
    if not root:
        return
    print(root.data, end=" ")
    self.printPreOrder(root.left)
    self.printPreOrder(root.right)

def printInOrder(self, root):
    if not root:
        return
    self.printInOrder(root.left)
    print(root.data, end=" ")
    self.printInOrder(root.right)
```

- Printing using tree traversals
 - Preorder
 - Inorder
- Can you implement Postorder?
- See In-class exercises #3 in the next slide...

Transform-and-Conquer Algorithms

AVLTree print using PostOrder



Task #3 for this week's Worksheet

- Create a method to the AVLTree class that prints out the elements using a depth-first post order tree traversal
 - Actually save a new version of the AVLTree class with your new method and create a main function to demonstrate your method call and result

Go to IDLE and try to program it
Save your program in a .py file and submit it in the appropriate delivery room



MERRIMACK COLLEGE

Transform-and-Conquer Algorithms

Third Example

```
# Create a tree node
class TreeNode(object):
    def __init__(self, data):
        self.data = data
        self.left = None
        self.right = None
        self.height = 1
```

printHelper(root)

AVL Trees

```
# Print the tree
def printHelper(self, currPtr, indent, last):
    if currPtr != None:
        print(indent, end="")
        if last:
            print("R----", end="")
            indent += "    "
        else:
            print("L----", end="")
            indent += "|    "
        print(currPtr.data)
        self.printHelper(currPtr.left, indent, False)
        self.printHelper(currPtr.right, indent, True)
```

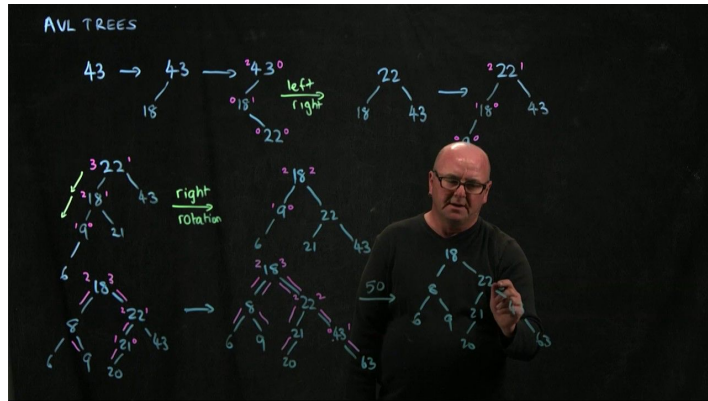


MERRIMACK COLLEGE

- Just a human readable print version to visualize the AVLTree (and yes, it is recursive)

Transform-and-Conquer Algorithms

Third Example



AVL Trees

- Now try it out the usage example downloading the avl.py file
 - <https://drive.google.com/file/d/1hSnE3w4fXmWfupeMFKhJTshU0f66Fes3/view?usp=sharing>
 - Feel free to edit it inserting and deleting elements to see the example at work
- The complexity of the insertion and deletion operations is $O(\log n)$
- Formal definition:
 - https://en.wikipedia.org/wiki/AVL_tree
- Implementation details:
 - <https://www.programiz.com/dsa/avl-tree>



Worksheet



Task #1 for this week's Worksheet

- Create a program that asks the user repeatedly positive integer numbers and stores/deletes it in an AVL tree (use the code seen in the third example)
 - *If the user enters a number already in the tree, delete it*
 - If the user enters a number that is not in the tree, insert it
 - After each insertion/deletion, the program prints the current tree using the printHelper method
 - When the user enters a non positive integer, the program ends
 - The entire program must run in logarithmic time.
- Besides the implementation of your program, write a short report describing your experiences.



Worksheet



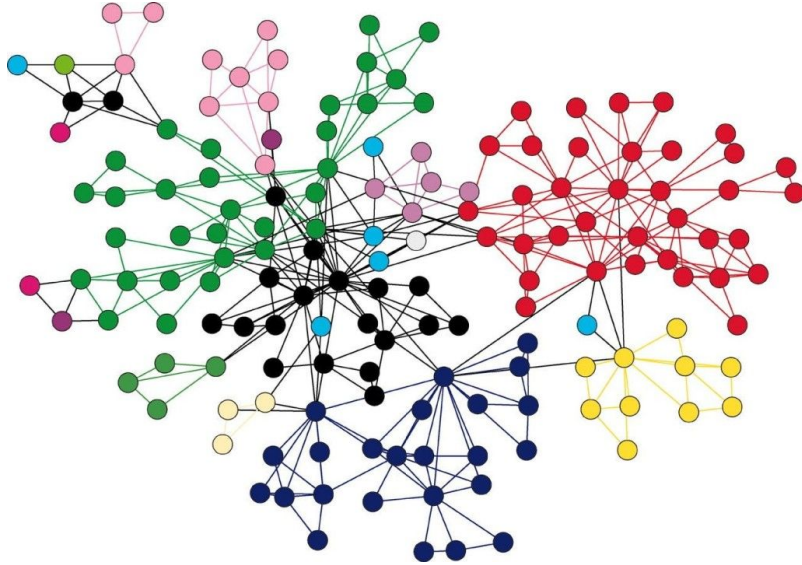
Task #1 for this week's Worksheet

- This program must be your own, do not use someone else's code
- Any specific questions about it, please bring to the Office hours meeting this Monday or contact me by email
- This is a challenging program to make sure you are mastering your Python programming skills, as well as your asymptotic analysis understanding
- Don't be shy with your questions

Go to IDLE and try to program it
Save your program in a .py file and submit it in the appropriate delivery room

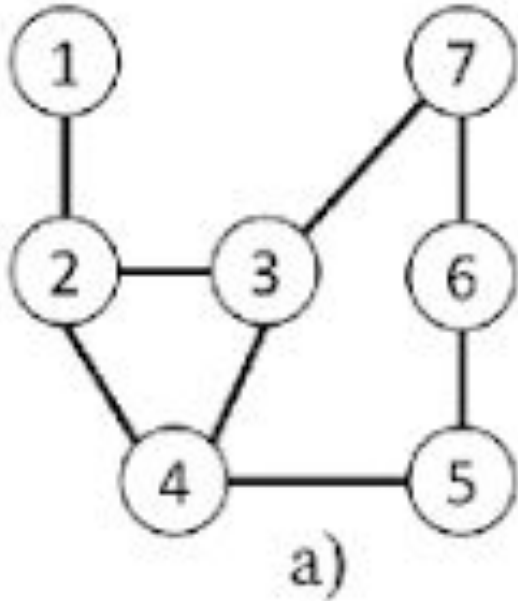


Graphs as Data Structures

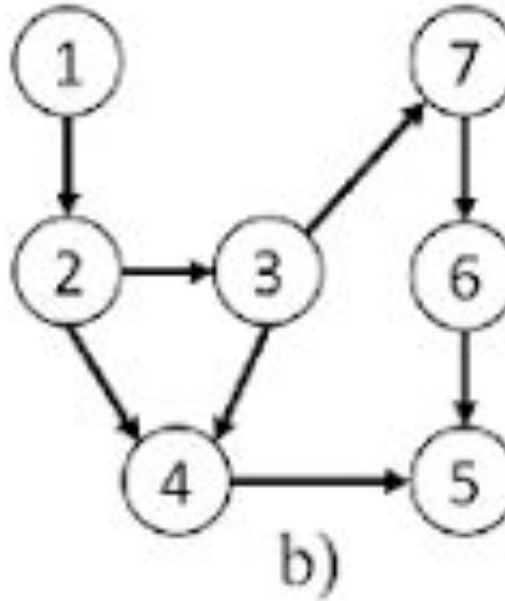


- Data structure which:
 - Consists of a finite set of nodes (or vertices)
 - Together with a set of unordered pairs (or edges) of these vertices for an undirected graph or a set of ordered pairs for a directed graph
 - In a weighted graph, a numeric value is attributed to each edge - for example representing the distance between two points in space

Types of Graphs: undirected vs directed

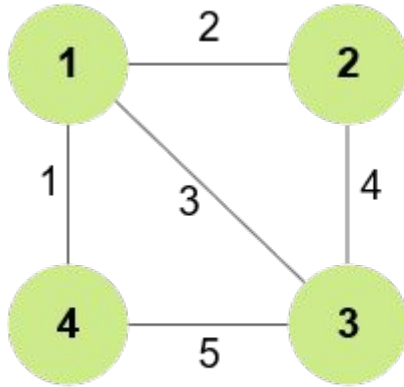


Undirected

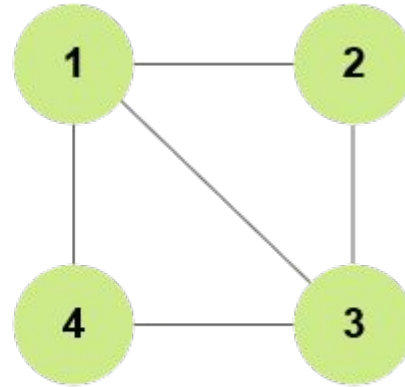


Directed

Types of Graphs: weighted vs unweighted



Weighted Graph



Unweighted Graph

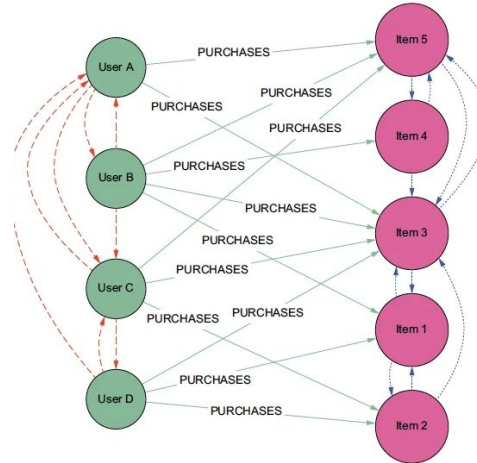
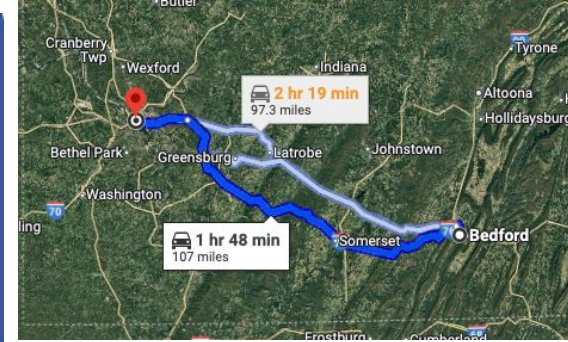
Common Applications for Graphs

Social Networks: Representing relationships between users, where nodes are users and edges are connections.

Web Page Linking: Modeling the internet as a graph, where web pages are nodes and hyperlinks are directed edges.

Routing Algorithms: Using graphs to find the shortest path in networks, such as in GPS navigation systems.

Recommendation Systems: Leveraging user-item interactions as a graph to suggest products or services.



Sources:
facebook.com
Google maps
graphaware.com

Common Problems for Graphs

Graph Traversal: Exploring nodes and edges systematically (e.g., Depth-First Search, Breadth-First Search).

Minimum Spanning Tree: Connecting all nodes with the minimum total edge weight (e.g., Prim's algorithms).

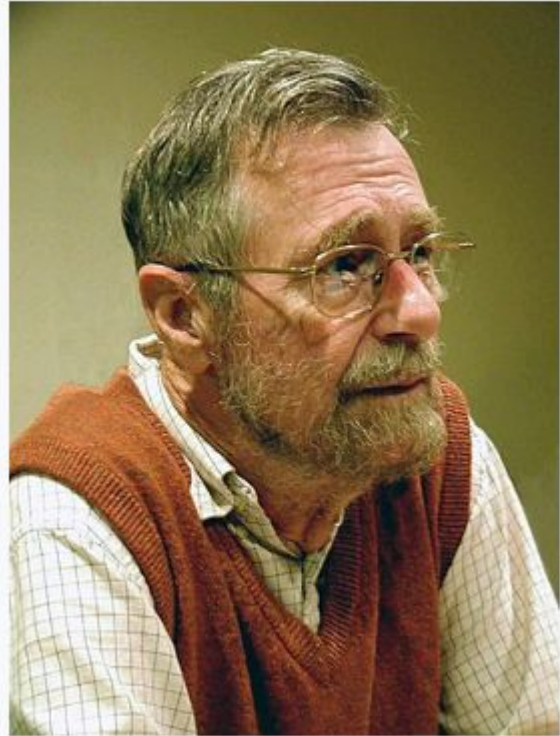
Cycle Detection: Identifying cycles in a graph, which is crucial for certain applications like deadlock detection.

Shortest Path: Finding the shortest route between nodes (e.g., Dijkstra's algorithm).

Dijkstra's Algorithm Background

- Proposed by computer scientist Edsger Dijkstra in late 1950s.
- Finds the shortest path from a source node to every other node
- Many applications

Edsger W. Dijkstra



Dijkstra's Algorithm Code

```
class Graph():  
  
    def __init__(self, vertices):  
        self.V = vertices  
        self.graph = [[0 for column in range(vertices)]  
                       | for row in range(vertices)]  
        self.paths = [[] for _ in range(self.V)]  
  
    def printSolution(self, dist):  
        print("Vertex \tDistance from Source \tPath")  
        for node in range(self.V):  
            print(node, "\t", dist[node], "\t\t\t", self.paths[node])
```

Dijkstra's Algorithm Code

```
# A utility function to find the vertex with
# minimum distance value, from the set of vertices
# not yet included in shortest path tree
def minDistance(self, dist, sptSet):

    # Initialize minimum distance for next node
    min = sys.maxsize

    # Search not nearest vertex not in the
    # shortest path tree
    for u in range(self.V):
        if dist[u] < min and sptSet[u] == False:
            min = dist[u]
            min_index = u

    return min_index
```


Dijkstra's Algorithm Code

```
def dijkstra(self, src):

    dist = [sys.maxsize] * self.V
    dist[src] = 0
    sptSet = [False] * self.V
    self.paths[src] = [src] # Initialize the path for the source

    for cout in range(self.V):

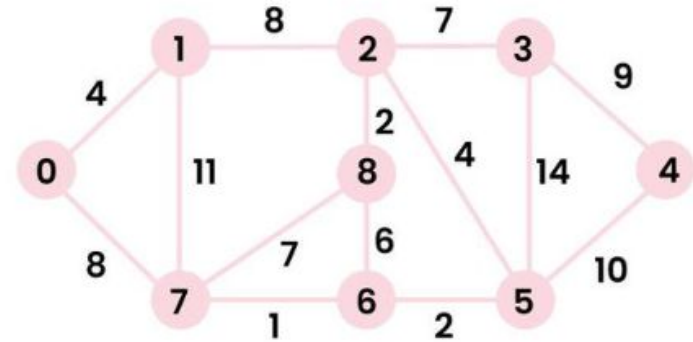
        # Pick the minimum distance vertex from
        # the set of vertices not yet processed.
        x = self.minDistance(dist, sptSet)
        # Put the minimum distance vertex in the
        # shortest path tree
        sptSet[x] = True

        # Update dist value of the adjacent vertices
        # of the picked vertex only if the current
        # distance is greater than new distance and
        # the vertex is not in the shortest path tree
        for y in range(self.V):
            if self.graph[x][y] > 0 and not sptSet[y] and \
               dist[y] > dist[x] + self.graph[x][y]:
                dist[y] = dist[x] + self.graph[x][y]
                self.paths[y] = self.paths[x] + [y] # Update the path

    self.printSolution(dist)
```

Dijkstra's Algorithm Example

```
if __name__ == "__main__":  
    g = Graph(9)  
    g.graph = [[0, 4, 0, 0, 0, 0, 0, 8, 0],  
               [4, 0, 8, 0, 0, 0, 0, 11, 0],  
               [0, 8, 0, 7, 0, 4, 0, 0, 2],  
               [0, 0, 7, 0, 9, 14, 0, 0, 0],  
               [0, 0, 0, 9, 0, 10, 0, 0, 0],  
               [0, 0, 4, 14, 10, 0, 2, 0, 0],  
               [0, 0, 0, 0, 0, 2, 0, 1, 6],  
               [8, 11, 0, 0, 0, 0, 1, 0, 7],  
               [0, 0, 2, 0, 0, 0, 6, 7, 0]]  
  
    g.dijkstra(0)
```



Vertex	Distance from Source	Path
0	0	[0]
1	4	[0, 1]
2	12	[0, 1, 2]
3	19	[0, 1, 2, 3]
4	21	[0, 7, 6, 5, 4]
5	11	[0, 7, 6, 5]
6	9	[0, 7, 6]
7	8	[0, 7]
8	14	[0, 1, 2, 8]

Transform-and-Conquer Algorithms

Programming Assignment



Project #7 - this week's Assignment

- Create a program that implements the Dijkstra's algorithm program with the Lord of the Rings example case
- You will need to represent the map of Middle Earth as an adjacency matrix and then feed it into the program
- Your program should print the shortest path from the Shire to various points in Middle Earth including the paths and the distances along the way.



Dijkstra's Algorithm Assignment: Lord of the Rings

Help Frodo plan a trip from the Shire to the crags of Mt. Doom so that he can destroy the One Ring and save Middle Earth.

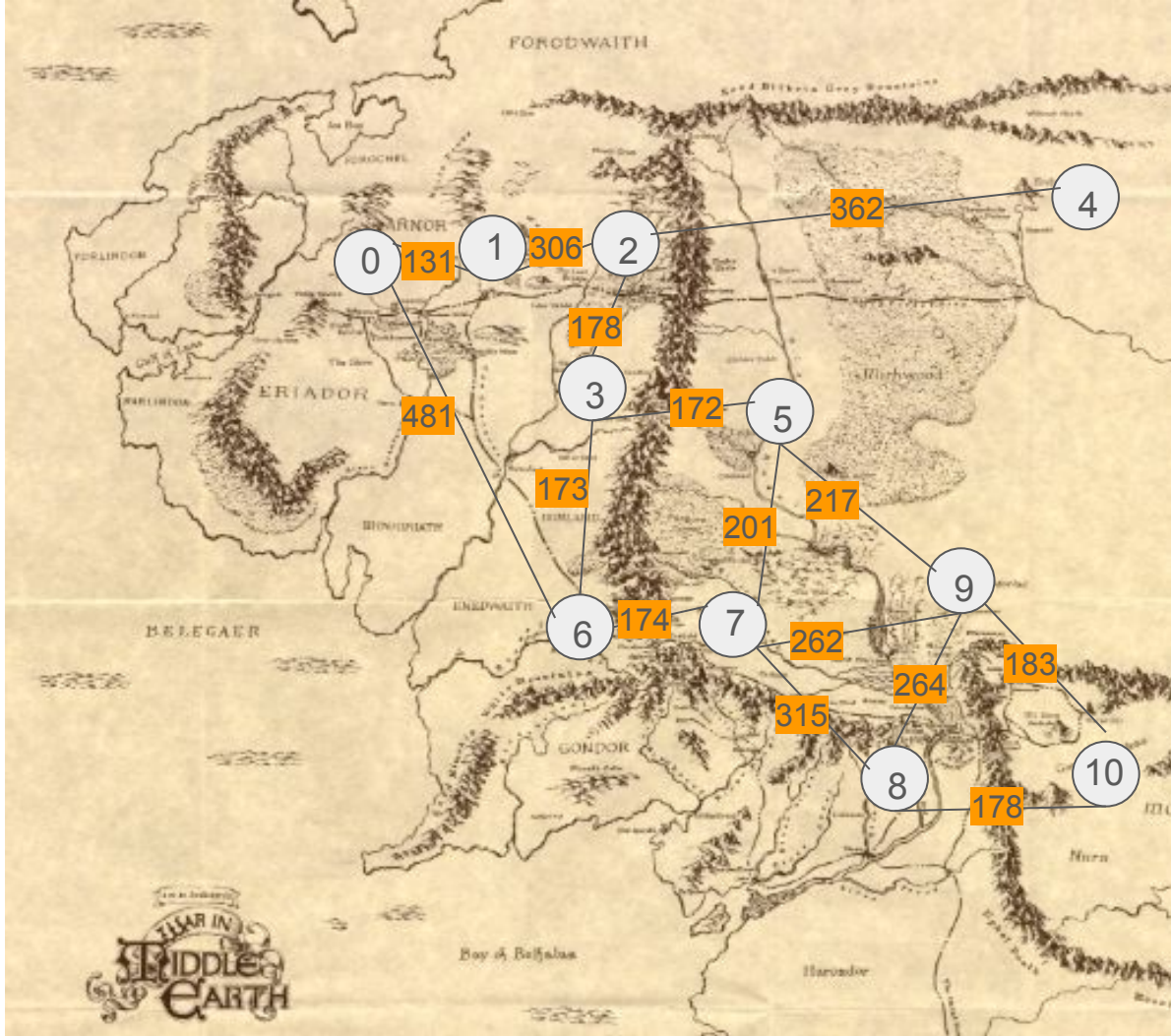


Dijkstra's Algorithm Assignment: Lord of the Rings

- Write a program that uses the Dijkstra's Algorithm program to determine the shortest paths from the Shire to other places Middle Earth.
- Specifically the program should determine the minimum distance Frodo would have to travel from the Shire to Rivendell, and then from Rivendell to Mt. Doom, so your program will have to run the algorithm twice.
- Your program should print out the results and the total amount Frodo needs to travel.

0. The Shire
1. Bree
2. Rivendell
3. Moria
4. Dale
5. Lorien
6. Isengard
7. Edoras
8. Minas Tirith
9. Eryn Muil
10. Mt. Doom

Note: path suggested by the algorithm may differ from actual path.



Programming Assignment



Project #2 - this week's Assignment

- This program must be your own, do not use someone else's code
- Any specific questions about it, please bring to the Office hours meeting this Monday or contact me by email
- This is a challenging program to make sure you are mastering your Python programming skills, as well as your asymptotic analysis understanding
- Don't be shy with your questions

Go to IDLE and try to program it
Save your program in a .py file and submit it in the appropriate delivery room



That's all for today folks!

This week's tasks

- Discussion: initial post by this Friday/replies by next Tuesday
- Tasks #1, #2, #3 and #4 for the In-class exercises
 - Deadline: Friday 11:59 PM EST
- Quiz #4 to be available this Friday
 - Deadline: Next Monday 11:59 PM EST
- Project #4 assignment
 - Deadline: Next Monday 11:59 PM EST
- Try all exercises seen in class and consult the reference sources, as the more you practice, the easier it gets



MERRIMACK COLLEGE

Have a Great Week!