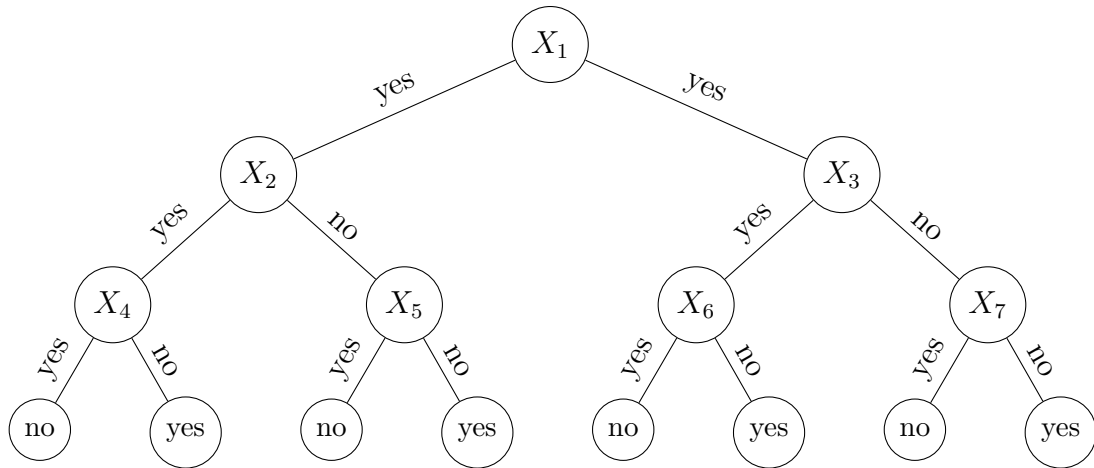
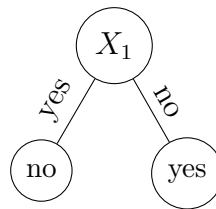


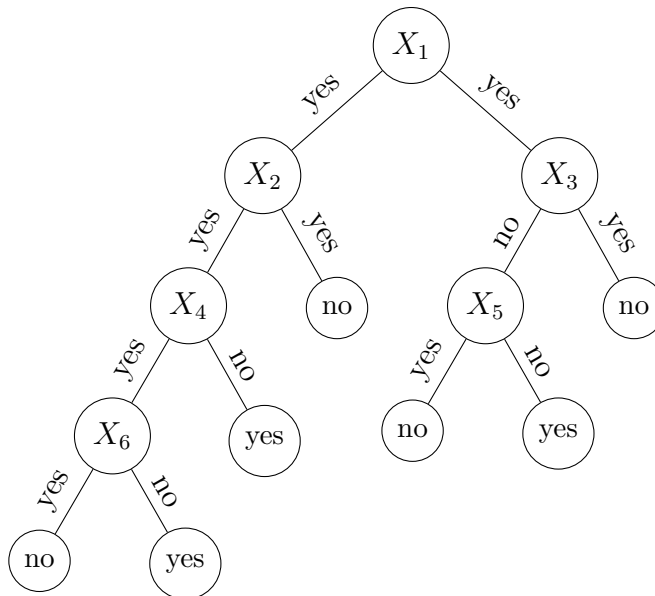
3. What is the depth of tree A? What is the depth of node X_4 in tree A?



4. What is the depth of tree B?



5. What is the depth of tree C? What are the depths of nodes X_1 and X_5 in tree C?



6. In-class coding and explanation of Depth First Traversal in Python.

Link to the code:

<https://colab.research.google.com/drive/1KypCp2tPDad4gdHjL1FH4DqbBnM5CfCr?usp=sharing>

Pre-order, Inorder and Post-order Tree Traversal

```
# This class represents an individual node
```

```
class Node:
```

```
    def __init__(self, key):
```

```
        self.left = None
```

```
        self.right = None
```

```
        self.val = key
```

```
def traversal1(root):
```

```
    if root is not None:
```

```
        # First recurse on left child
```

```
        traversal1(root.left)
```

```
        # then recurse on right child
```

```
        traversal1(root.right)
```

```
        # now print the data of node
```

```
        print(root.val, end='\t')
```

```
def traversal2(root):
```

```
    if root is not None:
```

```
        # First print the data of node
```

```
        print(root.val, end='\t')
```

```
        # Then recurse on left child
```

```
        traversal2(root.left)
```

```
        # Finally recurse on right child
```

```
        traversal2(root.right)
```

```
def traversal3(root):
```

```
    if root is not None:
```

```
        # First recurse on left child
```

```
        traversal3(root.left)
```

```
        # then print the data of node
```

```
        print(root.val, end='\t')
```

```
        # now recurse on right child
```

```
        traversal3(root.right)
```

```
def build_a_tree():
```

```
    root = Node(1)
```

```
    root.left = Node(2)
```

```
    root.right = Node(3)
```

```
    root.left.left = Node(4)
```

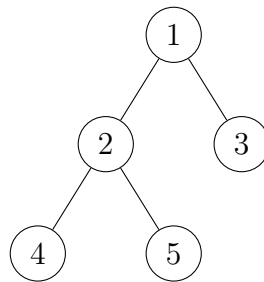
```
    root.left.right = Node(5)
```

```
    return root
```

```
if __name__ == '__main__':  
    root = build_a_tree()  
    print('traversal1 of the binary tree is: ')  
    traversal1(root)  
    print()  
    print('traversal2 of the binary tree is: ')  
    traversal2(root)  
    print()  
    print('traversal3 of the binary tree is: ')  
    traversal3(root)
```

Now, identify which traversal function is pre-order, in-order, post-order DFS:

- traversal1() is
- traversal2() is
- traversal3() is



Code Output

traversal1 of the binary tree is:

traversal2 of the binary tree is

traversal3 of the binary tree is

2 The Need For Speed: Vectorization and Numpy

Performing mathematical operations on vectors and matrices is ubiquitous in most machine learning algorithms. Whether it's a simple similarity measure that works by calculating the dot product between two vectors, or deep neural networks, they all involve repeated matrix operations. This makes it imperative that our underlying code design to perform matrix operations is efficient.

2.1 The Perils of Python

While Python is widely the language of choice for machine learning researchers across the globe (thanks to the speed of development and code readability it offers and the support it enjoys from the open-source community), Python as a high-level language on average is much slower than a lower level language like C++. To combat this, libraries like numpy and scipy implement most of the back-end operations they perform in C/C++, while providing wrappers in Python to be able to call underlying C code seamlessly from a Python script.

2.2 Speed Comparison: Numpy and Python

We highly recommend you to use *numpy* extensively in this course, it will be difficult to pass the programming portion of Homework 4 without writing most of your matrix operations in numpy. In this section, we'll see why.

Consider you have two vectors $\mathbf{a}, \mathbf{b} \in \mathbb{R}^n$. To see how similar they are, as measured by the cosine angle between them, you want to compute their dot product. This translates to the following operation:

$$\mathbf{a} \cdot \mathbf{b} = a_1b_1 + a_2b_2 + \dots + a_nb_n$$

When translated to code, notice how the dot product in NumPy is a whopping 100x faster than the native Python!

```
from timeit import timeit
import numpy as np
import array

VECTOR_SIZE = int(1e8)

# NumPy arrays
a = np.random.rand(VECTOR_SIZE)
b = np.random.rand(VECTOR_SIZE)

# Python arrays
aArr = array.array('d', a)
bArr = array.array('d', b)
```

```
def test_np():
    return np.dot(a, b)

# faster than multiprocessing, python lists, or numpy arrays with
# python loops
# faster than using a range and indexing
def test_py_arr():
    return sum(x * y for x, y in zip(aArr, bArr))

def time_dot_product(f):
    return timeit(f, setup=f, number=5) / 5

if __name__ == "__main__":
    print(f"NumPy = {time_dot_product(test_np):.2f}") # 0.05s
    print(f"Python on an array = {time_dot_product(test_py_arr):.2f}")
        # 5.45s
```

2.3 Useful Numpy Operations

Some operations in numpy that you will find really useful in your assignments are:

- [np.matmul](#): Matrix multiplication of two matrices
- [np.unique](#): Returns unique elements along an axis.
- [np.hstack](#): Stack two arrays horizontally (column-wise)
- [np.expand_dims](#): Convert a row vector of size n into a matrix of size $n * 1$ or $1 * n$
- [np.log](#), [np.sum](#), [np.exp](#), [@](#), [.T](#), and so on...

You can read [C vs. Python](#) for more details, and you can also read these two tutorials ([beginner](#), [intermediate](#)) from the official numpy website. For instance, understanding broadcasting is recommended. It will help you debug the shape errors you might face in all future homeworks.

3 ML Concepts: Construction of Decision Trees

In this section, we will go over how to construct our decision tree learner on a high level. The following questions will help guide the discussion:

1. What exactly are the tasks we are tackling?
2. What are the inputs and outputs at training time? At testing time?
3. At each node of the tree, what do we need to store?
4. What do we need to do at training time?
5. What do we need to do at testing time?
6. What happens if max depth is 0?
7. What happens if max depth is greater than the number of attributes?

