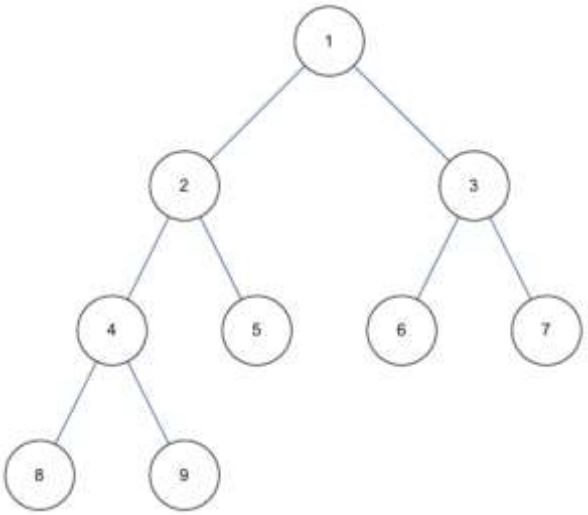
1. (5pt) Napisz dekorator, mający za zadanie
   - drukować informacje o czasie wykonywania funkcji
2. (5pt) Załóżmy, że mamy drzewo i reprezentujemy je na liście np. drzewo



reprezentujemy jako

```
["1", ["2", ["4", ["8", None, None], ["9",None,None]], ["5",None,None]], ["3", ["6", None, None],
["7", None, None]]]
```

Napisz funkcję która generuje w sposób losowy drzewo podanej wysokości oraz generator który przechodzi drzewo w porządku DFS i BFS.

3. (5pt) Napisz klasę **class Node(object)** do reprezentacji pojedynczego węzła drzewa z dowolną liczbą potomków. Podobnie jak w zadaniu poprzednim napisz funkcję która generuje losowo drzewo o danej wysokości i generator który przechodzi drzewo w porządku DFS i BFS.
4. (10pt) Przeciążenie funkcji (function overloading) daje możliwość wykorzystania tej samej nazwy funkcji, ale z różnymi parametrami. Na przykład w innych językach możemy napisać

```
float norm(float x, float y) {            // norma Euklidesowa
  return sqrt(x*x + y*y)
}
float norm(float x, float y, float z) {   // norma taksówkowa
  return abs(x) + abs(y) + abs(z)
}
```

W języku Python nie ma przeciążenia funkcji, po prostu następna definicja nadpisuje poprzednią. Napisz dekorator nazwijmy go **@overload**, który pozwala na taką własność. Przykładowy program powinien wyglądać tak

```
@overload
def norm(x,y):
  return math.sqrt(x*x + y*y)

@overload
def norm(x,y,z):
  return abs(x) + abs(y) + abs(z)

print(f"norm(2,4) = {norm(2,4)}")

print(f"norm(2,3,4) = {norm(2,3,4)}")
```

Otrzymujemy:

```
norm(2,4) = 4.47213595499958
norm(2,3,4) = 9
```

Wskazówka: Napisz dekorator, który wraca klasę z odpowiednio przeciążonym operatorem **__call__**, która przechowuje nazwy funkcji z parametrami. Do odróżnienia funkcji można wykorzystać np. **getfullargspec(f).args** z modułu **inspect** (from inspect import getfullargspec).