Multiple Dependency Pipeline: Takeaways

Syntax

- Building a DAG class:

```
class DAG:
    def __init__(self):
        self.root = Vertex()
    def add(self, node, to=None):
    if not node in self.graph:
        self.graph[node] = []
    if to:
        if not to in self.graph:
            self.graph[to] = []
        self.graph[node].append(to)
class Vertex:
    def __init__(self):
        self.to = []
        self.data = None
```

- Integrating a DAG into a pipeline:

```
class Pipeline:
    def __init__(self):
        self.tasks = DAG()
    def task(self, depends_on=None):
        def inner(f):
            self.tasks.add(f)
            if depends_on:
                self.tasks.add(depends_on, f)
        return f
    return inner
def run(self):
    scheduled = self.tasks.sort()
    completed = {}
    for task in scheduled:
        for node, values in self.tasks.graph.items():
            if task in values:
                completed[task] = task(completed[node])
        if task not in completed:
            completed[task] = task()
    return completed
```

Concepts

- A pipeline that handles multiple branching is called a Directed Acyclic Graph (DAG).
- Breaking down the terminology of a DAG:
    - Graph: A data structure that is composed of vertices and edges.
    - Directed: Each edge of a vertex points only in one direction.
    - Acyclic: The graph does not have any cycles, meaning that it cannot point to a vertex more than once.
- When using a DAG, we can implement task scheduling in linear time, $O(V+E)$, where V and E are the numbers of vertcies and edges.
- The time complexity for finding the longest path is $O(n^2)$ and $O(n \log n)$ for sorting by the longest paths.
- The number of in-degrees is what makes the root node different than any other node.
    - The number of in-degrees is the total count of edges pointing toward the node.
    - Each root node will always have zero in-degrees.
- A topological sort of a directed graph is a linear ordering of its vertices such that for every directed edge uv from vertex u to vertex v, u comes before v in the ordering.

Resources

- Deque module
- Kahn's Algorithm