In [1]:
```
from collections import defaultdict, deque

class Task:
    def __init__(self, name, duration):
        self.name = name
        self.duration = duration
        self.dependencies = []
        self.EST = 0
        self.EFT = 0
        self.LST = 0
        self.LFT = float('inf')

def add_dependency(tasks, from_task, to_task):
    tasks[from_task].dependencies.append(to_task)

def topological_sort(tasks):
    in_degree = {task: 0 for task in tasks}
    for task in tasks.values():
        for dep in task.dependencies:
            in_degree[dep] += 1

    zero_in_degree_queue = deque([task for task in tasks if in_degree[task] ==
    top_order = []

    while zero_in_degree_queue:
        current_task = zero_in_degree_queue.popleft()
        top_order.append(current_task)

        for dep in tasks[current_task].dependencies:
            in_degree[dep] -= 1
            if in_degree[dep] == 0:
                zero_in_degree_queue.append(dep)

    if len(top_order) == len(tasks):
        return top_order
    else:
        raise Exception("The graph has a cycle, so a topological sort is not p

def compute_earliest_times(tasks, top_order):
    for task_name in top_order:
        task = tasks[task_name]
        for dep_name in task.dependencies:
            dep = tasks[dep_name]
            dep.EST = max(dep.EST, task.EFT)
            dep.EFT = dep.EST + dep.duration

def compute_latest_times(tasks, top_order):
    end_time = max(task.EFT for task in tasks.values())
    for task in tasks.values():
        task.LFT = end_time

    for task_name in reversed(top_order):
        task = tasks[task_name]
        task.LST = task.LFT - task.duration
        for dep_name in task.dependencies:
            dep = tasks[dep_name]
            task.LFT = min(task.LFT, dep.LST)
```

```
def main():
    tasks = {
        'T_START': Task('T_START', 0),
        'A': Task('A', 3),
        'B': Task('B', 2),
        'C': Task('C', 4),
        'D': Task('D', 2),
        'E': Task('E', 3)
    }

    add_dependency(tasks, 'T_START', 'A')
    add_dependency(tasks, 'A', 'B')
    add_dependency(tasks, 'A', 'C')
    add_dependency(tasks, 'B', 'D')
    add_dependency(tasks, 'C', 'D')
    add_dependency(tasks, 'D', 'E')

    top_order = topological_sort(tasks)
    compute_earliest_times(tasks, top_order)
    compute_latest_times(tasks, top_order)

    for task in tasks.values():
        print("Task {}: EST = {}, EFT = {}, LST = {}, LFT = {}".format(task.na

    earliest_completion_time = max(task.EFT for task in tasks.values())
    latest_completion_time = max(task.LFT for task in tasks.values())

    print("Earliest time all tasks will be completed: {}".format(earliest_comp
    print("Latest time all tasks will be completed: {}".format(latest_completi

if __name__ == "__main__":
    main()
```

```
Task T_START: EST = 0, EFT = 0, LST = 12, LFT = 9
Task A: EST = 0, EFT = 3, LST = 9, LFT = 8
Task B: EST = 3, EFT = 5, LST = 10, LFT = 10
Task C: EST = 3, EFT = 7, LST = 8, LFT = 10
Task D: EST = 7, EFT = 9, LST = 10, LFT = 9
Task E: EST = 9, EFT = 12, LST = 9, LFT = 12
Earliest time all tasks will be completed: 12
Latest time all tasks will be completed: 12
```

In [ ]: