122022010 Shrayank Jai Mistry

Description: Write a program to check whether a given schedule is conflict serializable or not using precedence graph.

Programming Language: Python

```python
import networkx as nx
import matplotlib.pyplot as plt

class DiGraph:
    def __init__(self, vertices):
        self.adj_list = [set() for i in range(vertices)]
        self.vertices = vertices

    def add_edge(self, a, b):
        self.adj_list[a].add(b)

    def V(self):
        return self.vertices

    def adj(self, vertex):
        return list(self.adj_list[vertex])

    def create_graph(self, instrus_cnt, instrus):
        for i in range(instrus_cnt):
            for j in range(i + 1, instrus_cnt):
                t1 = instrus[i];
                t2 = instrus[j];

                if (t1[1] == t2[1]): continue
                if t1[-1] == t2[-1] and (t1[-2] == 'W' or t2[-2] == 'W'):
                    self.add_edge(int(t1[1]), int(t2[1]));

    def view_graph(self):
        for vertex in range(self.V()):
            print(vertex, ' = ', self.adj(vertex))
```

```python
class DirectedCycle:
    def __init__(self, graph):
        self.visited = [False for i in range(graph.V())]
        self.on_stack = [False for i in range(graph.V())]
        self.cycle = None
        self.edge_to =  [-1 for i in range(graph.V())]

        for vertex in range(graph.V()):
            if (not self.visited[vertex]):
                self.dfs(graph, vertex)

    def dfs(self, graph, v):
        self.on_stack[v] = True
        self.visited[v] = True

        for w in graph.adj(v):
            if (self.cycle): return
            if (not self.visited[w]):
                self.edge_to[w] = v
                self.dfs(graph, w)
            elif (self.on_stack[w]):
                self.cycle = list()
                x = v
                while (x != w):
                    self.cycle.append(x)
                    x = self.edge_to[x]
                self.cycle.append(w)
                self.cycle.append(v)
        self.on_stack[v] = False

    def has_cycle(self):
        return self.cycle != None

    def get_cycle(self):
        return self.cycle
```

```python
def show_graph(cycle, graph, flag):
    G = nx.DiGraph()
    edges = []
    for v in range(graph.V()):
        neighbs = graph.adj(v)
        for w in neighbs:
            temp = (v, w)
            edges.append(temp)
    # print(edges)

    G.add_nodes_from([node for node in range(graph.V())])
    G.add_edges_from(edges)
    pos = nx.circular_layout(G)
    plt.figure(figsize=(7, 7))

    ## Normal Nodes
    nx.draw_networkx_nodes(G, pos, nodelist=[v for v in range(graph.V())],
                           node_size=500, node_color='tomato', label='Non-Cycle Ts')

    ## Normal Edges
    nx.draw_networkx_edges(G, pos, edgelist=G.edges(),
                           connectionstyle='arc3, rad=0.09', edge_color='gray',
                           min_source_margin=0, min_target_margin=12)

    ## Cycle Nodes
    if (flag):
        nx.draw_networkx_nodes(G, pos, nodelist=cycle, node_size=500,
                               node_color='steelblue', label='Cycle Ts')

        start = 1
        N = len(cycle)
        cycle_edges = []

        while start > 0:
            cycle_edges.append((cycle[start - 1], cycle[start]))
            start = (start + 1) % N

         ## Cycle Edges
        nx.draw_networkx_edges(G, pos, edgelist=cycle_edges,
                               connectionstyle='arc3, rad=0.09', edge_color='black',
                               min_source_margin=0, min_target_margin=12)
        plt.title('Not Conflict Serilizable', loc='center', pad = 10)
    else:
        plt.title('Conflict Serilizable', loc='center', pad = 10)


    node_labels = dict()

    for node in range(graph.V()):
        node_labels[node] = 'T'+str(node)

    nx.draw_networkx_labels(G, pos, labels=node_labels, font_size=8)

    plt.legend(markerscale=0.5, loc="upper left", frameon=False)
    plt.savefig("graph_plot.jpg")
    plt.show()
```

```python
file = open("transaction_input.txt", "r")
data = []
for line in file:
    data.append(line.strip())

# print(data)

[transactionCnt, instructionCnt] = [int(data[0]), int(data[1])]
instructions = data[2:]

graph = DiGraph(transactionCnt)

# print(transactionCnt, instructionCnt, instructions)
graph.create_graph(instructionCnt, instructions)

# graph.view_graph()

di_cycle = DirectedCycle(graph)

# print(di_cycle.has_cycle())

if (di_cycle.has_cycle()):
    cycle = di_cycle.get_cycle()[::-1]
    show_graph(cycle, graph, True)
    print("Not Conflict Serializable")
    start = 1
    N = len(cycle)

    while start > 0:
        print('Edge from', 'T'+str(cycle[start - 1]), '--> T'+str(cycle[start]))
        start = (start + 1) % N
else:
    show_graph([], graph, False)
    print("Conflict Serializable")
```

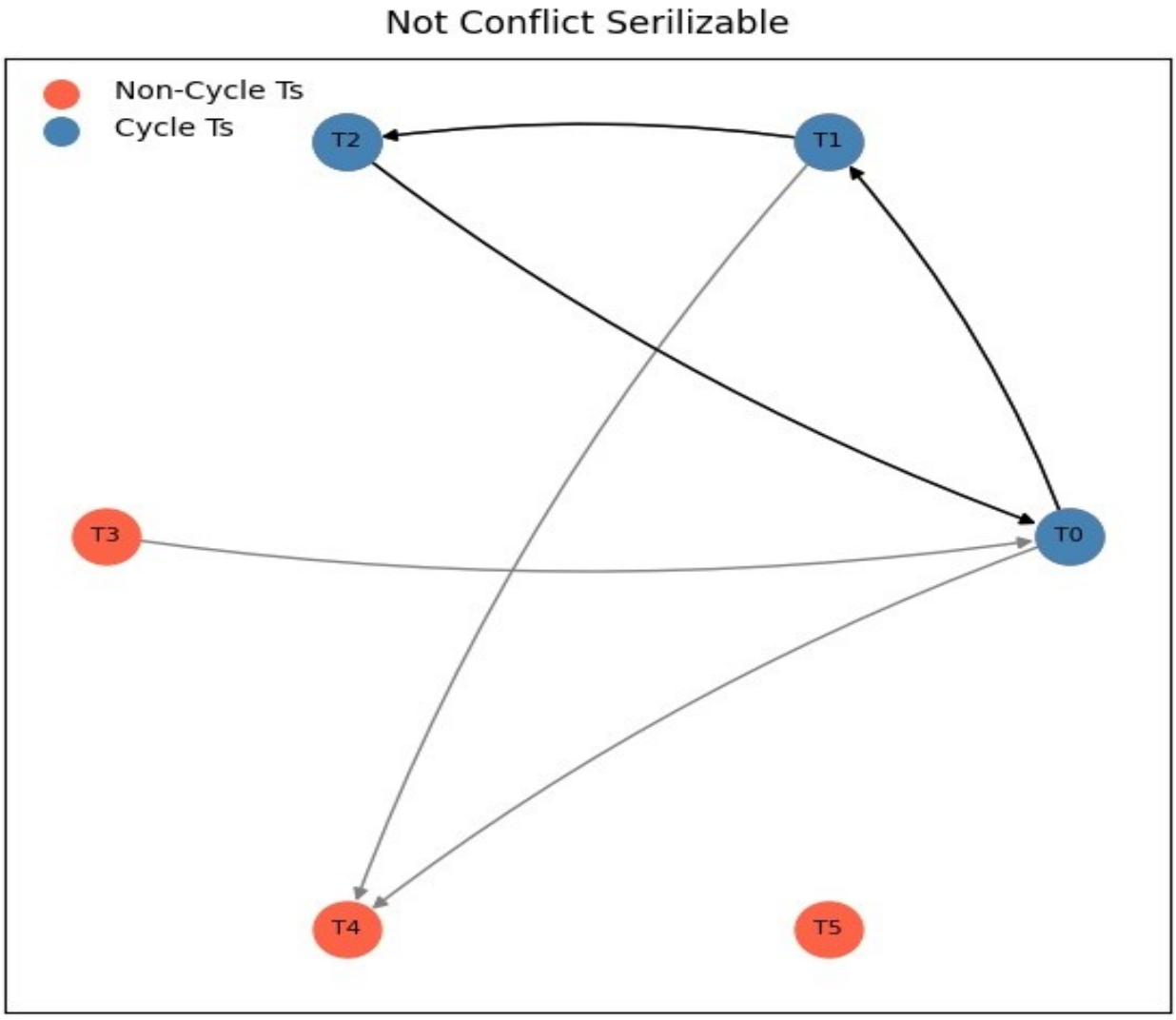
INPUT

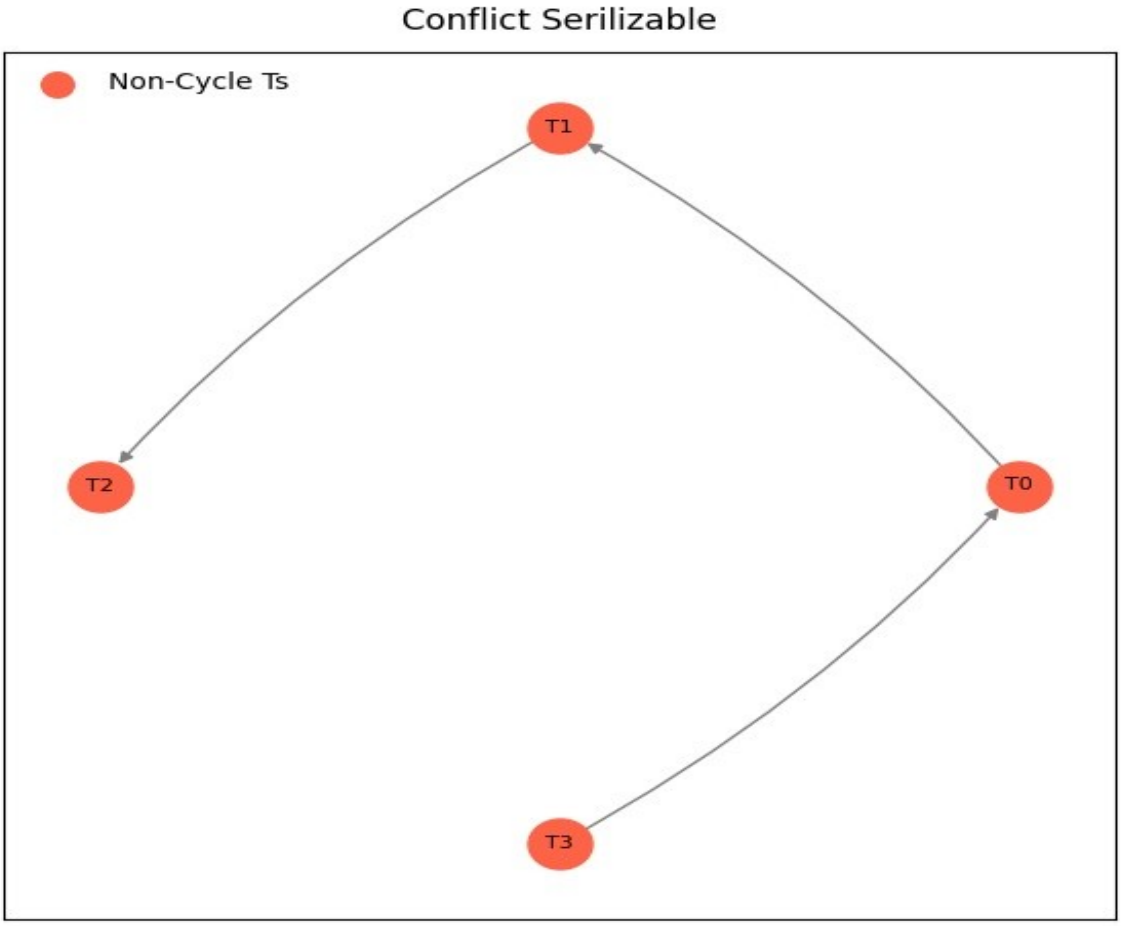
#* 6 10 [T0RA, T1WA, T1WB, T2WB, T2RC, T0WC, T3WD, T0WD, T4WA, T5RF] Not Conflict Serilizable
#* 4 6 [T0RA, T1WA, T1WB, T2WB, T3RC, T0WC] Conflict Serilizable

OUTPUT

#* jackson@ubuntu:~/GitHub/M.Tech-FY-Programs/Topics in Database Lab$ python3 directed_graph_cycle.py
#* Not Conflict Serializable
#* Edge from T2 --> T0
#* Edge from T0 --> T1
#* Edge from T1 --> T2

#* jackson@ubuntu:~/GitHub/M.Tech-FY-Programs/Topics in Database Lab$ python3 directed_graph_cycle.py
#* Conflict Serializable


Not Conflict Serilizable
Non-Cycle Ts
Cycle Ts
T2
T1
T3
T0
T4
T5


Conflict Serilizable
Non-Cycle Ts
T1
T2
T0
T3