|  |
| --- |
| **WEEK-10: Construct a dependency graph for the given syntax directed definition**  import networkx as nx  import matplotlib.pyplot as plt  # Create an empty directed graph  dependency\_graph = nx.DiGraph()  # Define the syntax-directed definition (SDD)  sdd = {      'E': [('E', '+', 'T'), ('E', '-', 'T'), ('T',)],      'T': [('T', '\*', 'F'), ('T', '/', 'F'), ('F',)],      'F': [('(', 'E', ')'), ('num',)]  }  # Add nodes and edges to the graph based on the SDD  for non\_terminal, productions in sdd.items():      for production in productions:          dependency\_graph.add\_node(non\_terminal)          for symbol in production:              dependency\_graph.add\_node(symbol)              dependency\_graph.add\_edge(non\_terminal, symbol)  # Draw the graph  pos = nx.spring\_layout(dependency\_graph)  nx.draw(dependency\_graph, pos, with\_labels=True, node\_size=2000, node\_color="skyblue", font\_size=12, font\_weight="bold")  plt.title("Dependency Graph for Syntax-Directed Definition")  plt.show()  **OUTPUT:**    **WEEK-11: Construct the DAG for the given 3 address code give a python code**  import networkx as nx  import matplotlib.pyplot as plt  # Sample three-address code  three\_address\_code = [  ('t1', '=', 'a', '+', 'b'),  ('t2', '=', 'c', '\*', 'd'),  ('t3', '=', 't1', '+', 't2'),  ('e', '=', 't3')  ]  # Create an empty Directed Acyclic Graph (DAG)  dag = nx.DiGraph()  # Dictionary to store node IDs for variables  variable\_nodes = {}  # Parse the three-address code and construct the DAG  for statement in three\_address\_code:  result\_var = statement[0]  operation = statement[1]  operands = statement[2:]  # Add result variable node to DAG  if result\_var not in variable\_nodes:  variable\_nodes[result\_var] = len(variable\_nodes)  dag.add\_node(variable\_nodes[result\_var], label=result\_var)  # Add operand variables and operation nodes to DAG  for operand in operands:  if operand not in variable\_nodes:  variable\_nodes[operand] = len(variable\_nodes)  dag.add\_node(variable\_nodes[operand], label=operand)  if operand != operation:  dag.add\_edge(variable\_nodes[result\_var], variable\_nodes[operand], label=operation)  # Draw the DAG  pos = nx.spring\_layout(dag)  nx.draw(dag, pos, with\_labels=True, node\_size=2000, node\_color="skyblue", font\_size=12, font\_weight="bold", arrowsize=20)  edge\_labels = nx.get\_edge\_attributes(dag, 'label')  nx.draw\_networkx\_edge\_labels(dag, pos, edge\_labels=edge\_labels)  plt.title("Directed Acyclic Graph (DAG) for Three-address Code")  plt.show() |