```
#Troy Krupinski
#tsk0064
import networkx as nx
import matplotlib.pyplot as plt
import numpy as np
G = nx.DiGraph()


#CSCE 5215 - Machine Learning
#Project 3 - Bayesian Networks

#Network:
'''
--------------------
*******************
Layer 4:


-------------------
car_won't_start:

Parents:[No_oil, Battery_flat]

children:[]

Co-parents:[]

Markov-blanket: [No_oil, Battery_flat]

-----------------
Dipstick:

Parents: [No_oil]

children: []

Co-parents:[]

Markov-blanket: [No_oil]

---------------------
Gas_gauge:

Parents: [Battery_flat]

Children: []


Co-parents:[]

Markov-blanket: [Battery_flat]

----------------------
Lights:

Parents: [Battery_flat]

Children: []

Co-parents:[]

Markov-blanket: [Battery_flat]
-----------------------

*********************
Layer 3:






-----------------------
Battery_flat:

Parents: [No_charging, Battery_dead]
```

```
Children: [car_won't_start, Gas_gauge, Lights]

Co-parents: [No_oil]

Markov-blanket: [No_charging, Battery_dead, car_won't_start, Gas_gauge, Lights, No_oil]

-------------------------

No_oil:

parents: []

children: [Dipstick, car_won't_start]

Co-parents: [Battery_flat]

Markov-blanket: [Dipstick, car_won't_start, Battery_flat]
--------------------


*********************
Layer 2:



----------------------
No_charging:

Parents: [Alternator_broken, Fanbelt_broken]

Children: [Battery_flat]

Co-parents: [Battery_dead]

Markov-blanket: [Alternator_broken, Fanbelt_broken, Battery_flat, Battery_dead]
---------------------

Battery_dead:

Parents: [Battery_age]

Children: [Battery_flat]

Co-parents: [No_charging]

Markov-blanket: [Battery_age, Battery_flat, No_charging]
---------------------


***********************
Layer 1:


-----------------------
Battery_age:

Parents: []

Children: [Battery_dead]

Co-parents: []

Markov-blanket: [Battery_dead]

-----------------------
Alternator_broken:

Parents: []

Children: [No_charging]

Co-parents: [Fanbelt_broken]

Markov-blanket: [No_charging, Fanbelt_broken]

-----------------------
```

```
Fanbelt_broken:

Parents: []

Children: [No_charging]

Co-parents: [Alternator_broken)

Markov-blanket: [No_charging, Alternator_broken]

'''

G.add_node("battery_age", ba_y=.2, ba_n = .8)
G.add_node("alternator_broken", ab_y=.1, ab_n = .9)
G.add_node("fanbelt_broken", fb_y = .3, fb_n = .7)
G.add_node("battery_dead", ba_y_bd_y = .7, ba_y_bd_n = .3, ba_n_bd_y = .3, ba_n_bd_n = .7)

G.add_node("no_charging_table",

           ab_y_fb_y_nc_y = .75,
           ab_y_fb_n_nc_y = .4,
           ab_n_fb_y_nc_y = .6,
           ab_n_fb_n_nc_y = .1,
           ab_y_fb_y_nc_n= 0.25,
           ab_y_fb_n_nc_n = .6,
           ab_n_fb_y_nc_n = .4,
           ab_n_fb_n_nc_n = .9)

G.add_node("battery_flat",
    bd_y_nc_y_bf_y=0.95, bd_y_nc_n_bf_y=0.85,
    bd_n_nc_y_bf_y=0.8, bd_n_nc_n_bf_y=0.1,
    bd_y_nc_y_bf_n=0.05, bd_y_nc_n_bf_n=0.15,
    bd_n_nc_y_bf_n=0.2, bd_n_nc_n_bf_n=0.9
)

G.add_node("no_oil", no_y=0.05, no_n=0.95)
G.add_node("lights",
    l_y_bf_y=0.9, l_n_bf_y=0.1,
    l_y_bf_n=0.3, l_n_bf_n=0.7
)
G.add_node("gas_gauge",
    bf_y_gg_y=0.1, bf_n_gg_y=0.95,
    bf_y_gg_n=0.9, bf_n_gg_n=0.05
)
G.add_node("car_wont_start",
    bf_y_no_y_cs_n=0.9, bf_y_no_n_cs_n=0.9,
    bf_n_no_y_cs_n=0.9, bf_n_no_n_cs_n=0.1,
    bf_y_no_y_cs_y=0.1, bf_y_no_n_cs_y=0.1,
    bf_n_no_y_cs_y=0.1, bf_n_no_n_cs_y=0.9
)

G.add_node("dipstick_low",
    no_y_dl_y=0.95, no_n_dl_y=0.3,
    no_y_dl_n=0.05, no_n_dl_n=0.7
)
edges = [
    ("battery_age", "battery_dead"),
    ("alternator_broken", "no_charging_table"),
    ("fanbelt_broken", "no_charging_table"),
    ("no_charging_table", "battery_flat"),
    ("battery_dead", "battery_flat"),
    ("battery_flat", "lights"),
    ("battery_flat", "gas_gauge"),
    ("battery_flat", "car_wont_start"),
    ("no_oil", "dipstick_low"),
    ("no_oil", "car_wont_start")

]

G.add_edges_from(edges)

plt.figure(figsize=(12, 8))

pos = nx.spring_layout(G, k=1, iterations=50)

nx.draw(G, pos, with_labels=True, node_size=3000, node_color="skyblue", font_size=10, font_weight="bold", font_color="black", edge_color="gra
```

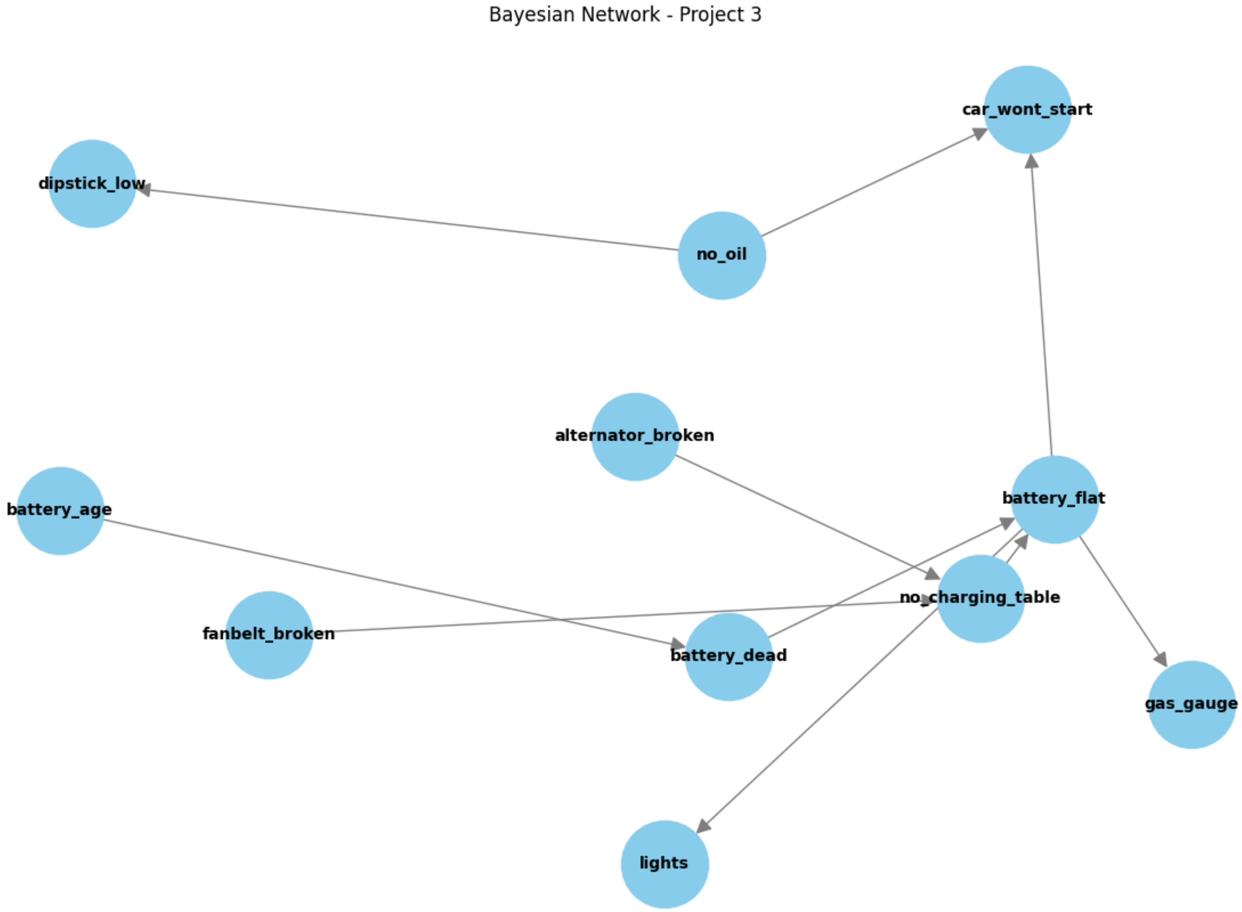
```
plt.title("Bayesian Network - Project 3")
plt.show()


fanbelt_probs = G.nodes["fanbelt_broken"]
battery_age_probs = G.nodes["battery_age"]
alternator_probs = G.nodes["alternator_broken"]


battery_dead_probs = G.nodes["battery_dead"]
no_charging_probs = G.nodes["no_charging_table"]


battery_flat_probs = G.nodes["battery_flat"]
no_oil_probs = G.nodes["no_oil"]

lights_probs = G.nodes["lights"]
gas_gauge_probs = G.nodes["gas_gauge"]
car_wont_start_probs = G.nodes["car_wont_start"]
dipstick_low_probs = G.nodes["dipstick_low"]

'''
P(B|+j, +m) =

P(B, e, a, +j, +m)
    P(+j, +m)
    Let us take P(B, e, a, +j, +m).
        Now P(B, e, a, +j, +m) = ∑e,a P(B, e, a, +j, +m) = ∑ P(B) × P(e) × P(a|B, e) × P(+j|a) × P(+m|a) e,a
        = P(B) × P(+e) × P(+a|B, +e) × P(+j|+a) × P(+m|+a) + P(B) × P(+e) × P(−a|B, +e) × P(+j|−a) ×
        P(+m|−a) + P(B) × P(−e) × P(+a|B, −e) × P(+j|+a) × P(+m|+a) + P(B) × P(−e) × P(−a|B, −e) ×
        P(+j|−a) × P(+m|−a)
'''

def car_fanbelt(G):
    """
    Calculate P(+cws|+fb) following the sum-product algorithm structure.
    For our case:
    P(+cws|+fb) = P(+cws,+fb) / P(+fb)
    where P(+cws,+fb) = ∑(bd,nc) P(+fb) × P(bd) × P(nc|+fb) × P(+cws|bd,nc)
    """
    # Root node probabilities
    fb_y = G.nodes["fanbelt_broken"]["fb_y"]  # P(+fb)

    # Get probabilities for battery dead from battery age
    ba_y = G.nodes["battery_age"]["ba_y"]
    ba_n = G.nodes["battery_age"]["ba_n"]
    ba_y_bd_y = G.nodes["battery_dead"]["ba_y_bd_y"]
    ba_n_bd_y = G.nodes["battery_dead"]["ba_n_bd_y"]

    # Get no charging probabilities given fanbelt
    ab_y = G.nodes["alternator_broken"]["ab_y"]
    ab_n = G.nodes["alternator_broken"]["ab_n"]
    ab_y_fb_y_nc_y = G.nodes["no_charging_table"]["ab_y_fb_y_nc_y"]
    ab_n_fb_y_nc_y = G.nodes["no_charging_table"]["ab_n_fb_y_nc_y"]

    # Get no oil probabilities
    no_y = G.nodes["no_oil"]["no_y"]
    no_n = G.nodes["no_oil"]["no_n"]

    # Get battery flat probabilities
    bd_y_nc_y_bf_y = G.nodes["battery_flat"]["bd_y_nc_y_bf_y"]
    bd_y_nc_n_bf_y = G.nodes["battery_flat"]["bd_y_nc_n_bf_y"]
    bd_n_nc_y_bf_y = G.nodes["battery_flat"]["bd_n_nc_y_bf_y"]
    bd_n_nc_n_bf_y = G.nodes["battery_flat"]["bd_n_nc_n_bf_y"]

    # Get car won't start probabilities
    bf_y_no_y_cs_y = G.nodes["car_wont_start"]["bf_y_no_y_cs_y"]
    bf_y_no_n_cs_y = G.nodes["car_wont_start"]["bf_y_no_n_cs_y"]
    bf_n_no_y_cs_y = G.nodes["car_wont_start"]["bf_n_no_y_cs_y"]
    bf_n_no_n_cs_y = G.nodes["car_wont_start"]["bf_n_no_n_cs_y"]

    # Calculate P(+cws,+fb) by summing over all combinations of intermediate variables
    # First, calculate P(bd) for both states
    p_bd_y = ba_y_bd_y * ba_y + ba_n_bd_y * ba_n  # P(+bd)
    p_bd_n = 1 - p_bd_y  # P(-bd)
```

```
    # Calculate P(nc|+fb) for both states
    p_nc_y_given_fb = ab_y_fb_y_nc_y * ab_y + ab_n_fb_y_nc_y * ab_n  # P(+nc|+fb)
    p_nc_n_given_fb = 1 - p_nc_y_given_fb  # P(-nc|+fb)

    # Now sum over all combinations of bd and nc states:
    # P(+cws,+fb) = ∑(bd,nc) P(+fb) × P(bd) × P(nc|+fb) × P(bf|bd,nc) × P(+cws|bf,no)
    p_cws_fb = 0

    # For battery dead = yes
    for bd_state in [(True, p_bd_y), (False, p_bd_n)]:
        bd_val, p_bd = bd_state
        # For no charging = yes/no
        for nc_state in [(True, p_nc_y_given_fb), (False, p_nc_n_given_fb)]:
            nc_val, p_nc = nc_state

            # Calculate P(bf|bd,nc)
            if bd_val and nc_val:
                p_bf_y = bd_y_nc_y_bf_y
            elif bd_val and not nc_val:
                p_bf_y = bd_y_nc_n_bf_y
            elif not bd_val and nc_val:
                p_bf_y = bd_n_nc_y_bf_y
            else:
                p_bf_y = bd_n_nc_n_bf_y

            p_bf_n = 1 - p_bf_y

            # Sum over battery flat states
            for bf_state in [(True, p_bf_y), (False, p_bf_n)]:
                bf_val, p_bf = bf_state
                # Sum over no oil states
                for no_state in [(True, no_y), (False, no_n)]:
                    no_val, p_no = no_state

                    # Get P(+cws|bf,no)
                    if bf_val and no_val:
                        p_cws = bf_y_no_y_cs_y
                    elif bf_val and not no_val:
                        p_cws = bf_y_no_n_cs_y
                    elif not bf_val and no_val:
                        p_cws = bf_n_no_y_cs_y
                    else:
                        p_cws = bf_n_no_n_cs_y

                    # Add this combination's contribution
                    p_cws_fb += fb_y * p_bd * p_nc * p_bf * p_no * p_cws

    # Finally, P(+cws|+fb) = P(+cws,+fb) / P(+fb)
    p_cws_given_fb = p_cws_fb / fb_y

    return p_cws_given_fb




def car_battery_age(G):
    """
    Calculate P(+cws|+ba) following the sum-product algorithm structure.
    For our case:
    P(+cws|+ba) = P(+cws,+ba) / P(+ba)
    """
    # Root node probabilities
    ba_y = G.nodes["battery_age"]["ba_y"]  # P(+ba)

    # Get probabilities for battery dead given battery age
    ba_y_bd_y = G.nodes["battery_dead"]["ba_y_bd_y"]  # P(+bd|+ba)

    # Get alternator and fanbelt probabilities (root nodes)
    ab_y = G.nodes["alternator_broken"]["ab_y"]
    ab_n = G.nodes["alternator_broken"]["ab_n"]
    fb_y = G.nodes["fanbelt_broken"]["fb_y"]
    fb_n = G.nodes["fanbelt_broken"]["fb_n"]

    # Get no charging probabilities for all combinations
    ab_y_fb_y_nc_y = G.nodes["no_charging_table"]["ab_y_fb_y_nc_y"]
    ab_y_fb_n_nc_y = G.nodes["no_charging_table"]["ab_y_fb_n_nc_y"]
    ab_n_fb_y_nc_y = G.nodes["no_charging_table"]["ab_n_fb_y_nc_y"]
```

```
    ab_n_fb_y_nc_y = G.nodes["no_charging_table"]["ab_n_fb_y_nc_y"]
    ab_n_fb_n_nc_y = G.nodes["no_charging_table"]["ab_n_fb_n_nc_y"]

    # Get no oil probabilities (root node)
    no_y = G.nodes["no_oil"]["no_y"]
    no_n = G.nodes["no_oil"]["no_n"]

    # Get battery flat probabilities
    bd_y_nc_y_bf_y = G.nodes["battery_flat"]["bd_y_nc_y_bf_y"]
    bd_y_nc_n_bf_y = G.nodes["battery_flat"]["bd_y_nc_n_bf_y"]
    bd_n_nc_y_bf_y = G.nodes["battery_flat"]["bd_n_nc_y_bf_y"]
    bd_n_nc_n_bf_y = G.nodes["battery_flat"]["bd_n_nc_n_bf_y"]

    # Get car won't start probabilities
    bf_y_no_y_cs_y = G.nodes["car_wont_start"]["bf_y_no_y_cs_y"]
    bf_y_no_n_cs_y = G.nodes["car_wont_start"]["bf_y_no_n_cs_y"]
    bf_n_no_y_cs_y = G.nodes["car_wont_start"]["bf_n_no_y_cs_y"]
    bf_n_no_n_cs_y = G.nodes["car_wont_start"]["bf_n_no_n_cs_y"]

    # Initialize probability sum
    p_cws_ba = 0

    # Given +ba, we know P(+bd|+ba)
    p_bd_y = ba_y_bd_y  # P(+bd|+ba)
    p_bd_n = 1 - p_bd_y  # P(-bd|+ba)

    # Sum over all possible combinations of alternator and fanbelt states
    for ab_state in [(True, ab_y), (False, ab_n)]:
        ab_val, p_ab = ab_state
        for fb_state in [(True, fb_y), (False, fb_n)]:
            fb_val, p_fb = fb_state

            # Calculate P(nc|ab,fb)
            if ab_val and fb_val:
                p_nc_y = ab_y_fb_y_nc_y
            elif ab_val and not fb_val:
                p_nc_y = ab_y_fb_n_nc_y
            elif not ab_val and fb_val:
                p_nc_y = ab_n_fb_y_nc_y
            else:
                p_nc_y = ab_n_fb_n_nc_y

            p_nc_n = 1 - p_nc_y

            # Sum over no charging states
            for nc_state in [(True, p_nc_y), (False, p_nc_n)]:
                nc_val, p_nc = nc_state

                # Calculate P(bf|bd,nc) for both battery dead states
                for bd_state in [(True, p_bd_y), (False, p_bd_n)]:
                    bd_val, p_bd = bd_state

                    # Get P(bf|bd,nc)
                    if bd_val and nc_val:
                        p_bf_y = bd_y_nc_y_bf_y
                    elif bd_val and not nc_val:
                        p_bf_y = bd_y_nc_n_bf_y
                    elif not bd_val and nc_val:
                        p_bf_y = bd_n_nc_y_bf_y
                    else:
                        p_bf_y = bd_n_nc_n_bf_y

                    p_bf_n = 1 - p_bf_y

                    # Sum over battery flat states
                    for bf_state in [(True, p_bf_y), (False, p_bf_n)]:
                        bf_val, p_bf = bf_state
                        # Sum over no oil states
                        for no_state in [(True, no_y), (False, no_n)]:
                            no_val, p_no = no_state

                            # Get P(+cws|bf,no)
                            if bf_val and no_val:
                                p_cws = bf_y_no_y_cs_y
                            elif bf_val and not no_val:
                                p_cws = bf_y_no_n_cs_y
                            elif not bf_val and no_val:
                                p_cws = bf_n_no_y_cs_y
```

```
                        else:
                            p_cws = bf_n_no_n_cs_y

                        # Add this combination's contribution
                        # P(+cws,+ba) = P(+ba) × P(bd|+ba) × P(ab) × P(fb) × P(nc|ab,fb) × P(bf|bd,nc) × P(+cws|bf,no) × P(no)
                        p_cws_ba += ba_y * p_bd * p_ab * p_fb * p_nc * p_bf * p_cws * p_no

    # Finally, P(+cws|+ba) = P(+cws,+ba) / P(+ba)
    p_cws_given_ba = p_cws_ba / ba_y

    return p_cws_given_ba

def calc_prob_alternator_given_lights_gasgauge(G):
    """
    Calculate P(+ab|-l,-gg) using Bayes' rule:
    P(+ab|-l,-gg) = P(-l,-gg|+ab)P(+ab) / P(-l,-gg)
    """
    # Get root probabilities
    ab_y = G.nodes["alternator_broken"]["ab_y"]  # P(+ab)
    ab_n = G.nodes["alternator_broken"]["ab_n"]  # P(-ab)
    fb_y = G.nodes["fanbelt_broken"]["fb_y"]  # P(+fb)
    fb_n = G.nodes["fanbelt_broken"]["fb_n"]  # P(-fb)
    ba_y = G.nodes["battery_age"]["ba_y"]  # P(+ba)
    ba_n = G.nodes["battery_age"]["ba_n"]  # P(-ba)

    # Get conditional probabilities
    # Battery dead given battery age
    ba_y_bd_y = G.nodes["battery_dead"]["ba_y_bd_y"]
    ba_n_bd_y = G.nodes["battery_dead"]["ba_n_bd_y"]

    # No charging given alternator and fanbelt
    ab_y_fb_y_nc_y = G.nodes["no_charging_table"]["ab_y_fb_y_nc_y"]
    ab_y_fb_n_nc_y = G.nodes["no_charging_table"]["ab_y_fb_n_nc_y"]
    ab_n_fb_y_nc_y = G.nodes["no_charging_table"]["ab_n_fb_y_nc_y"]
    ab_n_fb_n_nc_y = G.nodes["no_charging_table"]["ab_n_fb_n_nc_y"]

    # Battery flat given battery dead and no charging
    bd_y_nc_y_bf_y = G.nodes["battery_flat"]["bd_y_nc_y_bf_y"]
    bd_y_nc_n_bf_y = G.nodes["battery_flat"]["bd_y_nc_n_bf_y"]
    bd_n_nc_y_bf_y = G.nodes["battery_flat"]["bd_n_nc_y_bf_y"]
    bd_n_nc_n_bf_y = G.nodes["battery_flat"]["bd_n_nc_n_bf_y"]

    # Lights and gas gauge given battery flat
    l_n_bf_y = G.nodes["lights"]["l_n_bf_y"]  # P(-l|+bf)
    l_n_bf_n = G.nodes["lights"]["l_n_bf_n"]  # P(-l|-bf)
    bf_y_gg_n = G.nodes["gas_gauge"]["bf_y_gg_n"]  # P(-gg|+bf)
    bf_n_gg_n = G.nodes["gas_gauge"]["bf_n_gg_n"]  # P(-gg|-bf)

    # Calculate P(-l,-gg|+ab) and P(-l,-gg|-ab)
    p_evidence_given_ab_y = 0
    p_evidence_given_ab_n = 0

    # Sum over all possible paths
    for fb_state in [(True, fb_y), (False, fb_n)]:
        fb_val, p_fb = fb_state

        # Calculate P(nc|ab,fb)
        if fb_val:
            p_nc_y_given_ab_y = ab_y_fb_y_nc_y
            p_nc_y_given_ab_n = ab_n_fb_y_nc_y
        else:
            p_nc_y_given_ab_y = ab_y_fb_n_nc_y
            p_nc_y_given_ab_n = ab_n_fb_n_nc_y

        for ba_state in [(True, ba_y), (False, ba_n)]:
            ba_val, p_ba = ba_state

            # Calculate P(bd|ba)
            if ba_val:
                p_bd_y = ba_y_bd_y
            else:
                p_bd_y = ba_n_bd_y
            p_bd_n = 1 - p_bd_y

            for bd_state in [(True, p_bd_y), (False, p_bd_n)]:
                bd_val, p_bd = bd_state

                for nc_state in [(True, 1), (False, 0)]:  # We'll multiply by actual P(nc) later
```

```
                for nc_state in [(True, 1), (False, 0)]:  # We'll multiply by actual P(nc) later
                    nc_val, _ = nc_state

                    # Calculate P(bf|bd,nc)
                    if bd_val and nc_val:
                        p_bf_y = bd_y_nc_y_bf_y
                    elif bd_val and not nc_val:
                        p_bf_y = bd_y_nc_n_bf_y
                    elif not bd_val and nc_val:
                        p_bf_y = bd_n_nc_y_bf_y
                    else:
                        p_bf_y = bd_n_nc_n_bf_y
                    p_bf_n = 1 - p_bf_y

                    # Calculate P(-l,-gg|bf)
                    for bf_state in [(True, p_bf_y), (False, p_bf_n)]:
                        bf_val, p_bf = bf_state

                        if bf_val:
                            p_evidence = l_n_bf_y * bf_y_gg_n
                        else:
                            p_evidence = l_n_bf_n * bf_n_gg_n

                        # Add contribution to total probability
                        path_prob = p_fb * p_ba * p_bd * p_bf * p_evidence

                        if nc_val:
                            p_evidence_given_ab_y += path_prob * p_nc_y_given_ab_y
                            p_evidence_given_ab_n += path_prob * p_nc_y_given_ab_n
                        else:
                            p_evidence_given_ab_y += path_prob * (1 - p_nc_y_given_ab_y)
                            p_evidence_given_ab_n += path_prob * (1 - p_nc_y_given_ab_n)

    # Calculate P(-l,-gg) using total probability theorem
    p_evidence = p_evidence_given_ab_y * ab_y + p_evidence_given_ab_n * ab_n

    # Calculate P(+ab|-l,-gg) using Bayes' rule
    p_ab_given_evidence = (p_evidence_given_ab_y * ab_y) / p_evidence

    return p_ab_given_evidence


print("p(+cws|+fb)")
R2 = car_fanbelt(G)
print(f"R2: P(+cws | +fb): {R2}")
R3 = car_battery_age(G)
print(f"R3: p(+cws|+ba){R3}")
R4 = calc_prob_alternator_given_lights_gasgauge(G)
print(f"R4: p(+ab|-l,-gg){R4}")



print(G.nodes())
```




```
p(+cws|+fb)
R2: P(+cws | +fb): 0.34678719999999996
R3: p(+cws|+ba)0.32637360000000004
R4: p(+ab|-l,-gg)0.10940802798428095
['battery_age', 'alternator_broken', 'fanbelt_broken', 'battery_dead', 'no_charging_table', 'battery_flat', 'no_oil', 'lights', 'gas_gau
```