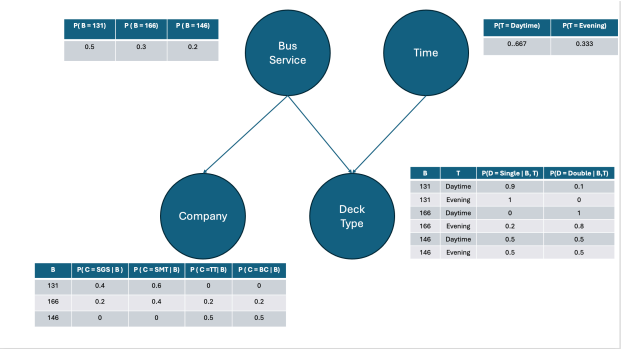
Question-1

(a)



(b)

P (D = Single, C = SMT | T = Evening)

= Σ P (D = Single, C =SMT, B| T = Evening)

Since B and T are independent without giving D

= Σ P (D = Single, C = SMT | T = Evening, B) P(B)

Since C and D are independent when given B,

=Σ P (D = Single | T = Evening, B) P (C = SMT | T = Evening, B) P(B)

=P (D = Single | T=Evening, B = 131) P (C = SMT | B =131) P (B =131) +

P (D = Single | T = Evening, B =166) P (C = SMT | B = 166) P(B=166) +

P (D = Single | T = Evening, B =146) P (C = SMT | B =146) P(B=146)

=1*0.6*0.5 + 0.2*0.4*0.3 + 0.5*0*0.2

=0.324

(c)

P (C =SMT | D = Single, T = Daytime)

= P (C = SMT, D = Single | T = Daytime) /P (D =Single | T = Daytime)

=Σ P (C = SMT, D = Single, B | T = Daytime) /Σ P (D = Single, B | T = Daytime)

=Σ P (B) P (C = SMT | B) P (D = Single | Daytime, B)/Σ P (D = Single | B, T = Daytime) P (B)

= (0.5* 0.6* 0.9 + 0.3*0.4*0 + 0.2*0*0.5) /(0.5*0.9 + 0.3*0 + 0.2*0.5)

= (0.27 + 0 + 0) /(0.45 + 0 + 0.1)

= 0.491

(d)

P (D = Double | B = 166)

= Σ P (D = Double, T | B =166)

= Σ P (D = Double | B = 166, T) P(T)

= 0.667*1 + 0.333*0.8

= 0.933

Question-2

(a)

```
# TE = Probability of being alive for the treated population - Probability of being alive for the untreated population.
# TE = P (dead = 0 , treatment = 1 )/P(treatment = 1) - P(dead = 0,treatment = 0)/P(treatment = 0)
#    = P (dead = 0 | treatment = 1) - P (dead = 0 | treatment = 0)

# get the number of alive people who get treated
num_treated_alive = len(df[(df['treatment'] == 1) & (df['dead'] == 0)])
cdp_treated_alive = num_treated_alive/num_treatment

# get the number of alive people who do not get treated
num_untreated_alive = len(df[(df['treatment'] == 0) & (df['dead'] == 0)])
cdp_untreated_alive = num_untreated_alive/num_untreatment

TE = cdp_treated_alive - cdp_untreated_alive
print('The TE only use the treatment and dead is:',TE)
Executed at 2024.05.25 15:46:09 in 29ms
```

```
The TE only use the treatment and dead is: -0.08484787082342737
```

The TE does not make sense because the total number of people do not get treated is twice as people who get treated.

But the number of smokers get treated is even larger than smokers do not get treated.

And we all known that smoking can significantly increase the chance of dead.

```
num_smoker_treatment = len(df[(df['smoker'] == 1) & (df['treatment'] == 1)])
num_smoker_untreatment = len(df[(df['smoker'] == 1) & (df['treatment'] == 0)])
print('The number of smoker get treated ', num_smoker_treatment)
print('The number of smoker did not get treated ', num_smoker_untreatment)
print('The number of people get treated ', num_treatment)
print('The number of people not get treated ', num_untreatment)
Executed at 2024.05.25 15:43:06 in 21ms
```

```
The number of smoker get treated  165919
The number of smoker did not get treated  134904
The number of people get treated  305775
The number of people not get treated  694225
```

(b)

```
print('The conditional prob of dead given treatment is : ')
print(factor_treated)
print('The conditional prob of dead given untreatment is :')
print(factor_untreated)
```

Executed at 2024.05.25 17:33:22 in 4ms

```
The conditional prob of dead given treatment is :
+---------+-------------+
| Dead    |   phi(Dead) |
+=========+=============+
| Dead(0) |      0.6792 |
+---------+-------------+
| Dead(1) |      0.3208 |
+---------+-------------+
The conditional prob of dead given untreatment is :
+---------+-------------+
| Dead    |   phi(Dead) |
+=========+=============+
| Dead(0) |      0.7640 |
+---------+-------------+
| Dead(1) |      0.2360 |
+---------+-------------+
```

```
print('The TE of Bob"s idea is :',factor_treated.values[0]-factor_untreated.values[0])
```

Executed at 2024.05.25 17:35:16 in 2ms

```
The TE of Bob"s idea is : -0.08484787082342737
```

Which is the same as question(a)

(c)

```
print('The conditional prob of dead given treatment is :')
print(factor_treated)
print('The conditional prob of dead given untreatment is :')
print(factor_untreated)
```

Executed at 2024.05.25 17:13:23 in 5ms

```
+=========+=============+
| Dead(0) |      0.7726 |
+---------+-------------+
| Dead(1) |      0.2274 |
+---------+-------------+
The conditional prob of dead given untreatment is :
+---------+-------------+
| Dead    |   phi(Dead) |
+=========+=============+
| Dead(0) |      0.6896 |
+---------+-------------+
| Dead(1) |      0.3104 |
+---------+-------------+
```

```
print('The TE of Alan"s idea is :',factor_treated.values[0]-factor_untreated.values[0])
```

Executed at 2024.05.25 17:13:51 in 3ms

```
The TE of Alan"s idea is : 0.08304460735461672
```

(d)

The fundamental difference between Bob and Alan’s Bayesian Network is:

In Bob’s BN, smoking habits will influence the probability of treatment, which means the but in Alan’s BN, smoking and treatment are conditional independent.

Bob’s BN should be more accurate because when we are calculating TE, we use treatment and dead only.

Question-3

(a)

Code for X-distance:

```
# (a)
def calculateXDistance(current, goal):
    row_count = 0
    column_count = 0
    for i in range(3):
        for j in range(3):
            if current.tiledArr[i][j] != -1:
                if current.tiledArr[i][j] != goal.tiledArr[i][j]:
                    goal_i, goal_j = numpy.where(goal.tiledArr == current.tiledArr[i][j])
                    if goal_i[0] != i:
                        row_count += 1
                    if goal_j[0] != j:
                        column_count += 1
    return row_count + column_count
```

Code for A* Search:

```
def a_star(initial_state, goal_state, heuristic):
    open_list = []
    closed_list = []
    open_list.append((initial_state, heuristic(initial_state, goal_state)))
    node_reopen_count = 0

    while open_list:
        current_state, current_cost = open_list.pop(0)

        if current_state.isEqual(goal_state):
            retrievePathFromState(current_state)
            print(f"Node reopen count: {node_reopen_count}")
            return current_state.gVal, node_reopen_count

        closed_list.append(current_state)

        for action in range(4):
            operator = Operator(action)
            next_state = current_state.applyOperator(operator)

            if isPresentStateInList(next_state, closed_list):
                continue

            next_cost = next_state.gVal + heuristic(next_state, goal_state)

            if isPresentStateInPriorityList(next_state, open_list):
                checkAndUpdateStateInPriorityQueue(open_list, next_state, next_cost)
                node_reopen_count += 1
            else:
                insertStateInPriorityQueue(open_list, next_state, next_cost)

    return None, node_reopen_count
```

Case1:

Hamming distance:

```
*******
Size of path is  22 22
Node reopen count: 861
Hamming distance - Cost: 22, Node reopens: 861
```

Manhattan distance:

```
Size of path is  22 22
Node reopen count: 42
Manhattan distance - Cost: 22, Node reopens: 42
```

X-distance:

```
Size of path is  22 22
Node reopen count: 140
X-distance - Cost: 22, Node reopens: 140
```

Case 2:

Hamming distance:

```
Size of path is  26 26
Node reopen count: 8251
Hamming distance - Cost: 26, Node reopens: 8251
```

Manhattan distance

```
Size of path is  26 26
Node reopen count: 302
Manhattan distance - Cost: 26, Node reopens: 302
```

X-distance:

```
Size of path is  26 26
Node reopen count: 1331
X-distance - Cost: 26, Node reopens: 1331
```

Case 3:

Hamming distance:

```
Size of path is  26 26
Node reopen count: 8235
Hamming distance - Cost: 26, Node reopens: 8235
```

Manhattan distance:

```
Size of path is  26 26
Node reopen count: 202
Manhattan distance - Cost: 26, Node reopens: 202
```

X-distance:

```
Size of path is  26 26
Node reopen count: 1238
X-distance - Cost: 26, Node reopens: 1238
```

(b)

Manhattan Distance: Often the best heuristic due to its balance between informativeness and efficiency.

X-Distance: A good middle ground, better than Hamming but not as effective as Manhattan.

Hamming Distance: The simplest but least effective, leading to higher costs and more node reopens.

Question-4

(a)

Since it's a 5 * 5 grid, and here are two walls occupy 3 grid and 2 grid respectively.

5 * 5 – 3 – 2 = 20 states.

(b)

$Q(S,A)=R(S,A,S')+\gamma V(S')$ { $R(S,A,S')$ = 1while S'= (5,3) . $R(S,A,S')$ = -1while $S' \neq$ (5,3)}

(c)

Iteration 0:

0	0	0	0	0
0		0		0
0		0		0
0		0	0	0
0	0	0	0	0

Iteration 1:

Since all values are initialized to 0:

$V(s)=-1+\max\{0.9\cdot 0+0.05\cdot 0+0.05\cdot 0\}=-1$

However, for the cell with the reward:

$V(3,5)=1+\max\{0.9\cdot 0+0.05\cdot 0+0.05\cdot 0\}=1$

-1	-1	-1	-1	-1
-1		-1		-1
-1		-1		1
-1		-1	-1	-1
-1	-1	-1	-1	-1

Iteration 2:

$V(2,5) = -1 + \max\{0.9 \cdot 1 + 0.05 \cdot (-1) + 0.05 \cdot (-1)\} = -0.2$

$V(4,5) = -1 + \max\{0.9 \cdot 1 + 0.05 \cdot (-1) + 0.05 \cdot (-1)\} = -0.2$

For other states:

$V(s) = -1 + \max\{0.9 \cdot (-1) + 0.05 \cdot (-1) + 0.05 \cdot (-1)\} = -2$

-2	-2	-2	-2	-2
-2		-2		-0.2
-2		-2		1
-2		-2	-2	-0.2
-2	-2	-2	-2	-2

(d)

States is determined by the positions of agent and pizza.

And here are 20 positions for agent and pizza, 20 * 20 =400 states in all.

(e)

The Bellman Equation accounts for the probabilistic nature of state transitions, which fits the dynamic environment where the agent might not always land in the desired grid.

Since we got 20 * 20 =400 states according to the question(d), and agent can choose any grid to jump which means he have 20 choices of actions, so here will be 20 * 20 * 20 = 8000 entries.