ECE 561 Hw 3

1a) $P(A^c, B^c) \overset{?}{=} P(A^c) \overset{?}{=} P(B) \rightarrow P(A^c) = P(A^c, B) + P(A^c, B^c)$

$P(A^c, B^c) = P(A^c, B^c) + P(A^c, B) \overset{?}{=} P(B)$

$0 = P(A^c, B) \overset{?}{=} P(B)$ $\boxed{\text{False}}$

$P(B) = P(A^c, B)$

1b) $P(A^c \cap B^c) \geq P(A^c) - P(B) \rightarrow P(A^c, B^c) \geq P(A^c) - P(B)$

$P(A^c \cap B^c) = P(A^c) + P(B^c) - P(A^c \cup B^c)$

$\cancel{P(A^c)} + P(B^c) - P(A^c \cup B^c) \geq \cancel{P(A^c)} - P(B) \rightarrow 1 \geq P(A^c \cup B^c)$

$\boxed{\text{True}}$

1c) $P(A) - P(B) \leq P(B^c) - P(A^c) \rightarrow P(A) + P(A^c) \leq P(B) + P(B^c)$

we know for any set S $P(S) + P(S^c) = 1$

so $1 \leq 1$ $\boxed{\text{True}}$

2a) All outcomes: G: gold, P: player, O: open

GPO, GOP, PGO, POG, OPG, OGP

2b) We find that staying at the originally chosen door yields a win rate of $\boxed{1/3}$ while switching to the remaining door yields a win rate of $\boxed{1/2}$

3a) $P(t=\text{cat}) = 1/2$ $P(t=\text{dog}) = 1/4$ $P(t=\text{fish}) = 1/4$

t = true p = predicted

$P(p=\text{cat}) = \frac{1}{2}(1-\varepsilon) + \frac{1}{4}(\varepsilon)$ $P(p=\text{dog}) = \frac{1}{4}(1-\varepsilon) + \frac{1}{2}(\varepsilon)$

$P(p=\text{fish}) = \frac{1}{4}(1-\varepsilon) + \frac{1}{4}(\varepsilon)$

$P(p=\text{cat}) = \frac{1}{2} - \frac{\varepsilon}{4}$ $P(p=\text{dog}) = \frac{1}{4} + \frac{\varepsilon}{4}$

$P(p=\text{fish}) = \frac{1}{4}$

3b) dog is predicted

$$P(t=dog \mid p=dog) = \frac{P(p=dog \mid t=dog) \cdot P(t=dog)}{P(p=dog)}$$

$$P(t=cat \mid p=dog) = \frac{P(p=dog \mid t=cat) \cdot P(t=cat)}{P(p=dog)}$$

$$P(t=fish \mid p=dog) = \frac{P(p=dog \mid t=fish) \cdot P(t=fish)}{P(p=dog)}$$

$$P(t=dog \mid p=dog) = \frac{(1-\varepsilon)(1/4)}{(1+\varepsilon)/4} = \boxed{\frac{1-\varepsilon}{1+\varepsilon}}$$

$$P(t=cat \mid p=dog) = \frac{\varepsilon \cdot (1/2)}{(1+\varepsilon)/4} = \boxed{\frac{2\varepsilon}{1+\varepsilon}}$$

$$P(t=fish \mid p=dog) = \boxed{0}$$

In [1]:
```
# ECE 561 - Jackson Hellmers
import numpy as np
```

In [2]:
```
'''
Generates all the outcomes in the set of rolling a 6 sided die 7 times
'''
def gen_trial():
    set_list = set()
    current_set = ""
    count = 0
    gen_trial_helper(set_list,current_set,1,count)
    gen_trial_helper(set_list,current_set,2,count)
    gen_trial_helper(set_list,current_set,3,count)
    gen_trial_helper(set_list,current_set,4,count)
    gen_trial_helper(set_list,current_set,5,count)
    gen_trial_helper(set_list,current_set,6,count)
    return set_list
```

In [3]:
```
'''
Helper method for gen_trial function
'''
def gen_trial_helper(set_list,current_set,to_add,count):
    if count == 7:
        set_list.add(current_set)
        return
    current_set = current_set+str(to_add)
    count+=1
    gen_trial_helper(set_list,current_set,1,count)
    gen_trial_helper(set_list,current_set,2,count)
    gen_trial_helper(set_list,current_set,3,count)
    gen_trial_helper(set_list,current_set,4,count)
    gen_trial_helper(set_list,current_set,5,count)
    gen_trial_helper(set_list,current_set,6,count)
```

In [4]:
```
'''
Function to add multiple values to the same key in a dictionary
'''
def add_values_in_dict(sample_dict, key, list_of_values):
    if key not in sample_dict:
        sample_dict[key] = list()
    sample_dict[key].extend(list_of_values)
    return sample_dict
```

In [5]:
```
# Part A

total_set = gen_trial() # Generate all outcomes in set and verify its size
assert (len(total_set) == (6**7)),"Size of set is incorrect"
print("Size of set:")
print(len(total_set))
```

```
Size of set:
279936
```

In [6]:
```
# Part B

# Since the dice is fair and each toss is independent all outcomes
# are equiprobable.
print("Probability of Each Equiprobable Outcome:")
print(str(100/len(total_set))+"%")
```

```
Probability of Each Equiprobable Outcome:
0.00035722450845907634%
```

In [7]:
```
# Part C and D
```

```python
# Get number of times each side of the die appears in the outcome
# Use this as the key to the dictionary
multinomial_dict = dict()
for outcome in total_set:
    to_add = ""
    to_add += str(outcome.count('1')) # For example is outcomes was 1145366
    to_add += str(outcome.count('2')) # the key would be 201112
    to_add += str(outcome.count('3')) # as there are two 1s, zero 2s,
    to_add += str(outcome.count('4')) # one 3, one 4, one 5, and two 6s
    to_add += str(outcome.count('5'))
    to_add += str(outcome.count('6'))
    add_values_in_dict(multinomial_dict,to_add,[outcome]) # Add outcome and its key to dict

# Our set is 6 different integer values which can take the value of {0,1,2,3,4,5,6,7}
# the 6 integers must add up to 7
```

In [8]:

```python
# Part E

print("Number of Different Multinomial Outputs")
print(len(multinomial_dict.keys()))
```

```
Number of Different Multinomial Outputs
792
```

In [9]:

```python
# Part F

num_outputs = np.array([len(x) for x in multinomial_dict.values()]) # Convert key and values into l
key_list = list(multinomial_dict.keys())
sorted_index = np.argsort(num_outputs,axis=None) # Sort by keys with most values
print(num_outputs[sorted_index[-10:]])
```

```
[1260 1260 1260 1260 2520 2520 2520 2520 2520 2520]
```

In [10]:

```python
# Part F continued

largest_indices = sorted_index[-6:] # Get indices of largest dictionary keys
print("Most Probable Multinomial Outputs (Each Index Shows Number of Occurances For That Label):")
for index in largest_indices:
    print(key_list[index]) # Print out most common dictionary keys

prob = num_outputs[largest_indices[0]]/num_outputs.sum() # Calculate probability
print("Probability of Most Probable Outputs (6 Outputs Have Equiprobably Max Probability):")
print(str(round(prob*100,4))+"%")

# We see that the most probable outcomes are when every label occurs at least once
# with one of the labels occuring twice. Which conceptually makes sense
```

```
Most Probable Multinomial Outputs (Each Index Shows Number of Occurances For That Label):
111112
111211
112111
121111
211111
111121
Probability of Most Probable Outputs (6 Outputs Have Equiprobably Max Probability):
0.9002%
```

In [ ]: