CS 1.2: Intro to Data Structures & Algorithms

Histogram & Markov Chain Worksheet

Name: _____

Text: *"I like dogs and you like dogs. I like cats but you hate cats."* (ignore all punctuation)

Histograms

Q1: How many distinct word *types* are present in this input text? How many total word *tokens*?

Distinct word types: ____ Total word tokens: ____

Q2: What data structure would be appropriate to store a histogram counting *word frequency*? Why did you choose this data structure? In other words, what makes this data structure ideal?

Q3: Write the data structure you would create to store this histogram counting *word frequency* (as it would look if you printed it out with Python).

Markov Chains

Q4: Draw a conceptual diagram of the *Markov chain* generated from analyzing the text above. Label each *state transition arc* with the count of how many times you observed that word pair.

Q5: Write the data structure you would create to store the word transitions out of the state that represents the word "*like*" in this Markov chain (as it would look if you printed it out with Python).

Q6: Write a *new sentence* that can be *generated* by doing a *random walk* on this Markov chain.

CS 1.2: Intro to Data Structures & Algorithms

Sampling Worksheet

Name: _____

Sampling

Assume you have a histogram data structure that stores the following words and counts:

```
histogram = [('cats', 3), ('dogs', 4), ('rabbits', 2), ('turtles', 1)]
```

Review the code below that implements a non-uniform sampling function given a histogram:

```
def sample(histogram):
    """Return a word from this histogram, randomly sampled by weighting
    each word's probability of being chosen by its observed frequency."""
    tokens = sum([count for word, count in histogram])   # Count total tokens
    dart = random.randint(1, tokens)    # Throw a dart on the number line
    # Note: Assume that randint returns 8 here and dart stores the value 8
    fence = 0              # Border of where each word splits the number line
    for word, count in histogram:    # Loop over each word and its count
        fence += count     # Move this word's fence border to the right
        if fence >= dart:  # Check if this word's fence is past the dart
            return word    # Fence is past the dart, so choose this word
```

Q7: Execute the code above as the Python interpreter would. Complete the table below to keep track of the value of each variable inside the `for` loop. Write "N/A" if a value is never evaluated.

Iteration	word	count	fence	dart	fence >= dart
1	'cats'	3	3	8	False
2				8	
3				8	
4	'turtles'	1		8	

Q8: Which `word` is returned when the `sample` function is executed? (Assume `dart`'s value is `8`.)

Q9: Mark the number line below to show each `word`'s `count` and `fence` values from the table above and where the value of `dart` is on the number line to determine which `word` is returned: