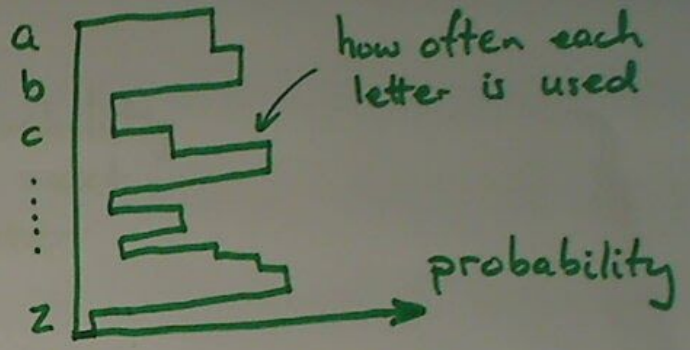
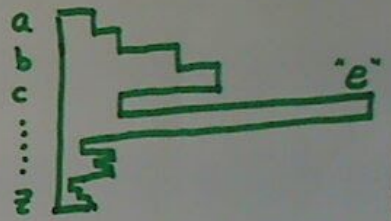
FRIDAY, WEEK 10

# Modelling a Sequence



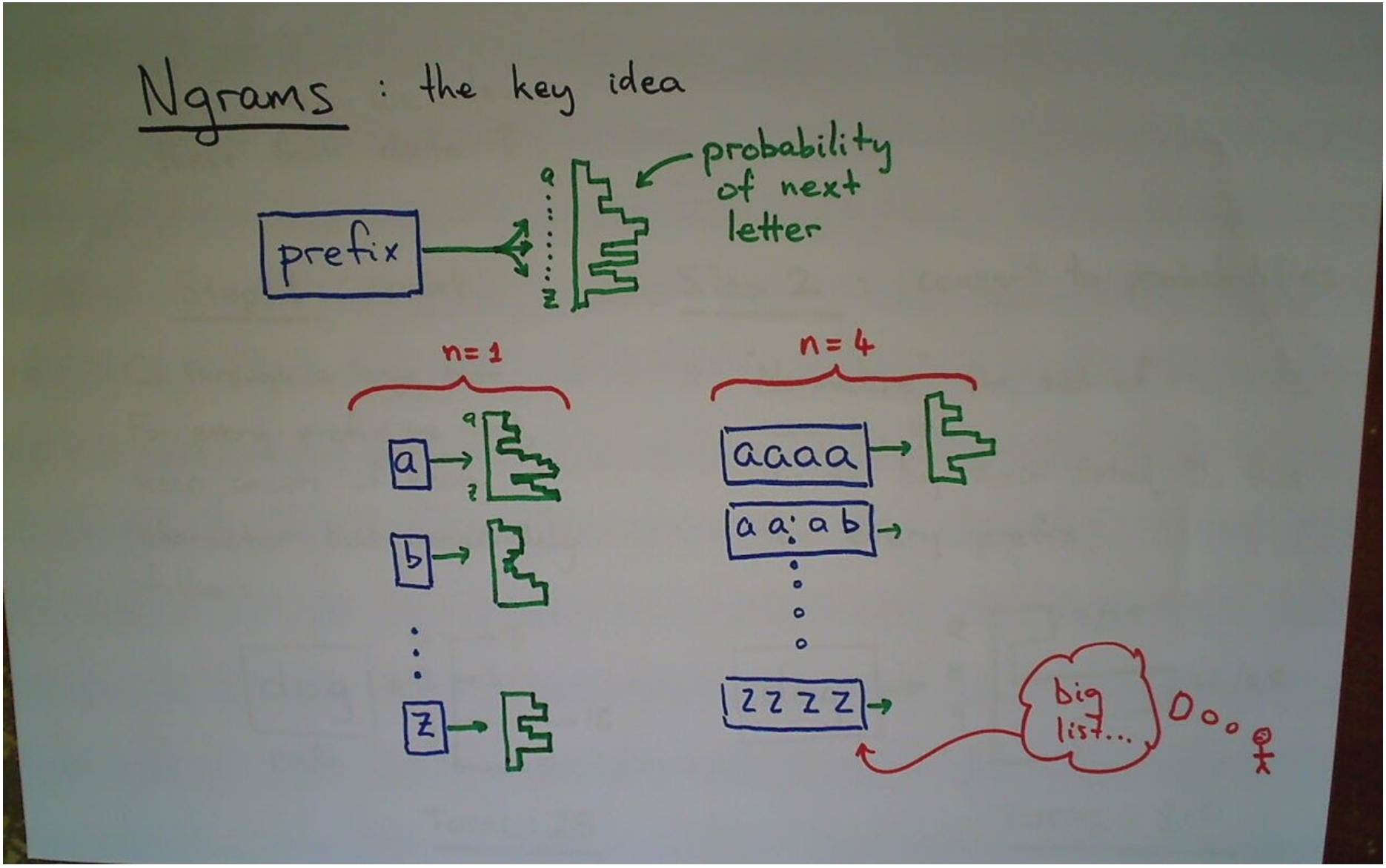

Grammar models overall structure
- → compliance (yes/no)
- → parse tree

Letter-by-letter model, with probabilities
- → predictions
- → how well something fits a pattern




"Once upon a time, ther ?"

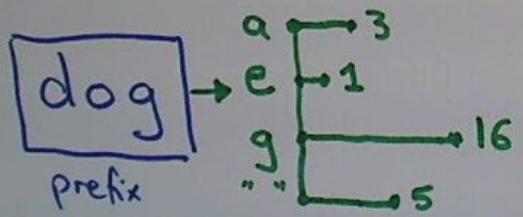
# Ngrams : the key idea



How can we learn these from data?

## Step 1: count

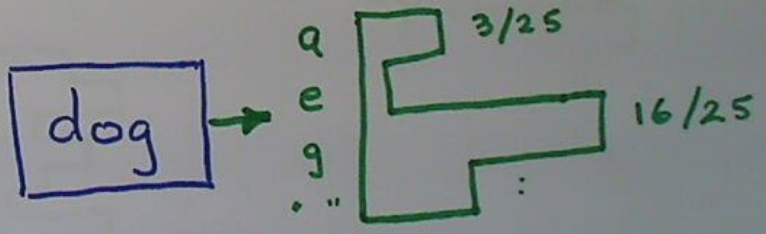
Go through a large text. For every prefix we meet, keep counts of every character that immediately follows




TOTAL: 25

## Step 2: convert to probabilities

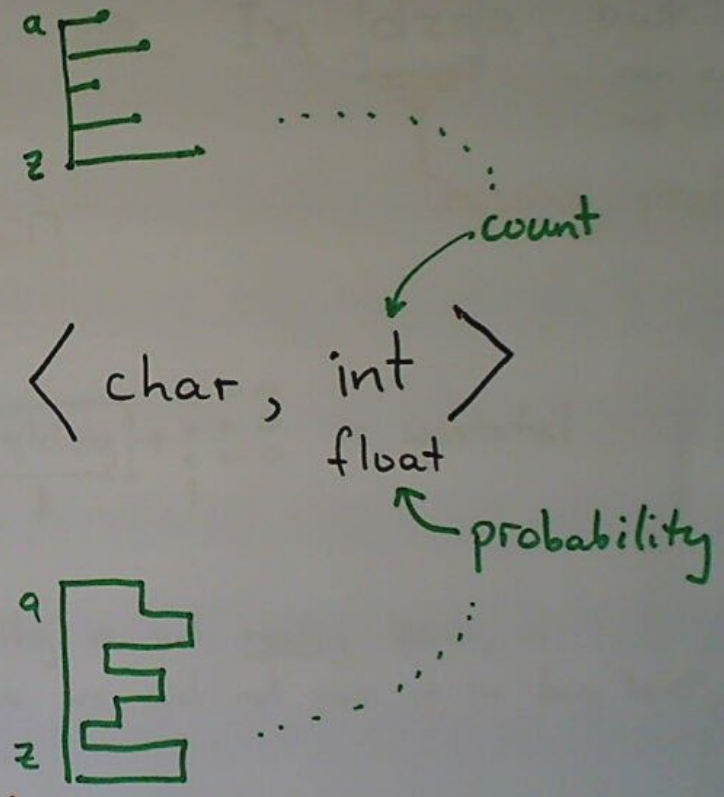
"Normalise" the set of counts, ie. divide by their total. Do this for every prefix.



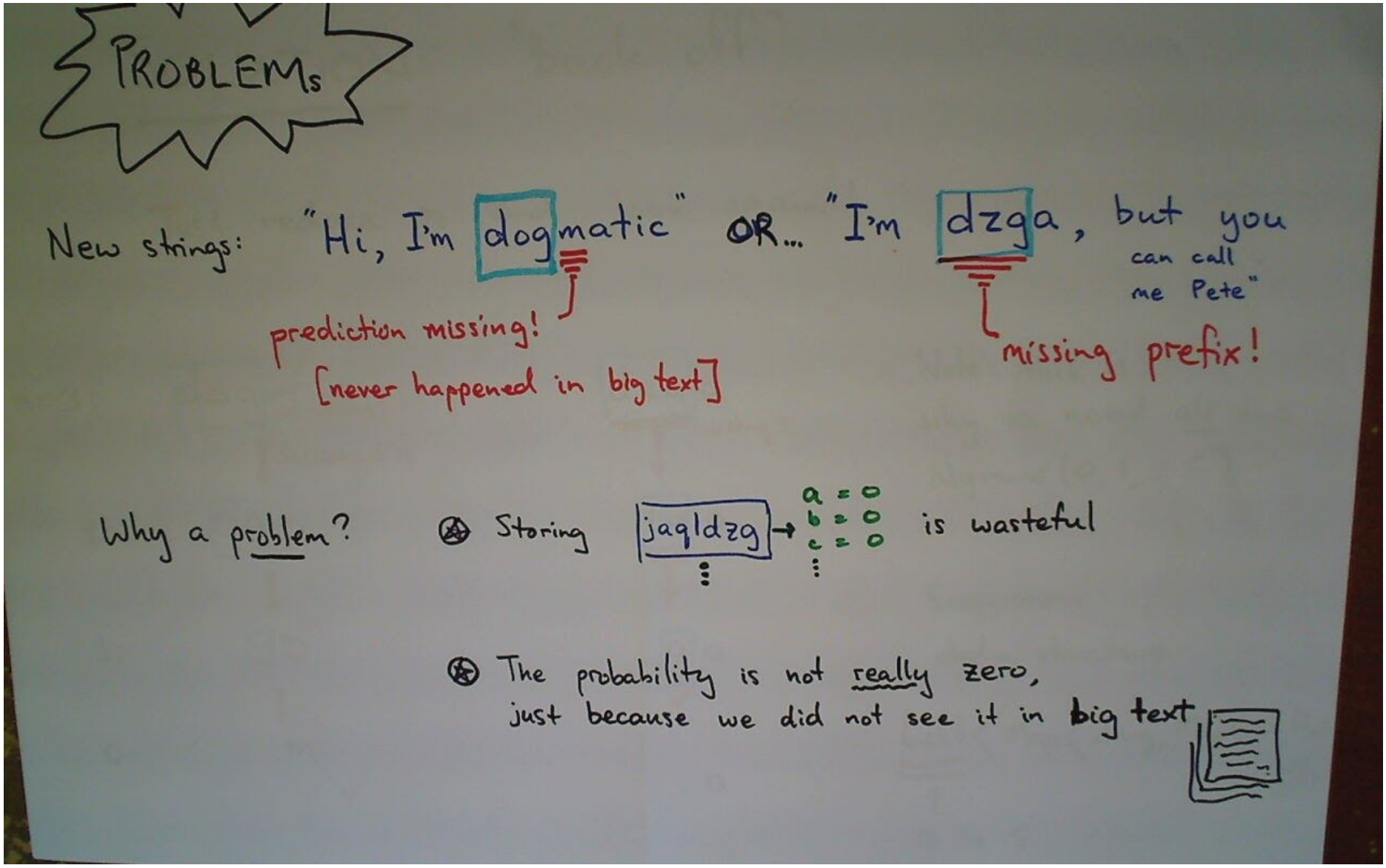

TOTAL: 1.0

# Suggestion

for data structure:

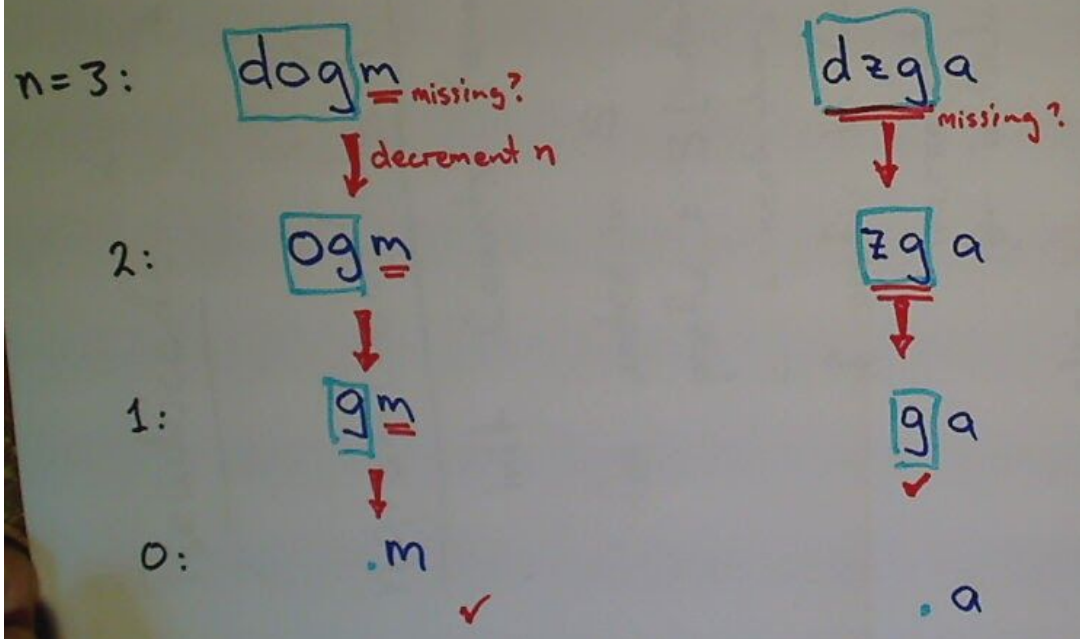



Notice we only store what we have seen ≥ once.


PROBLEMs
New strings: "Hi, I'm dogmatic" OR... "I'm dzga, but you can call me Pete"
prediction missing!
[never happened in big text]
missing prefix!
Why a problem?
Storing jaqldzg → a = 0 b = 0 c = 0
is wasteful
The probability is not really zero, just because we did not see it in big text

# SOLUTION: "back off"

Just reduce n and look again!




Note: this is why we need all the Ngrams [0, 1, .. 5].

Suggestion: data structure

List<Map<String, Map<char, float>>>

↑

0 to 5

Pseudocode ?

a big big string

"n"grams... (size of prefix)

ngramProbsCalc ( S , n )

```
init Counts (a map)

for index in S :
    prefix = S[index to index + n]
    c = next char, at S[index + n]
    if prefix in Counts :
        increment relevant entry
        (or initialise new entry c → 1)
    else : //brand new prefix...
        new key in Counts,
        value is a Map,
        set entry in that  c → 1

initialise Probs (a map)
    for each key in Counts :
        calc sum total
        Divide by total and put in Probs
```