12.25

Input

boy
cow
bow
⋮

} translate

269
269
269
⋮

count how many times each occurs.

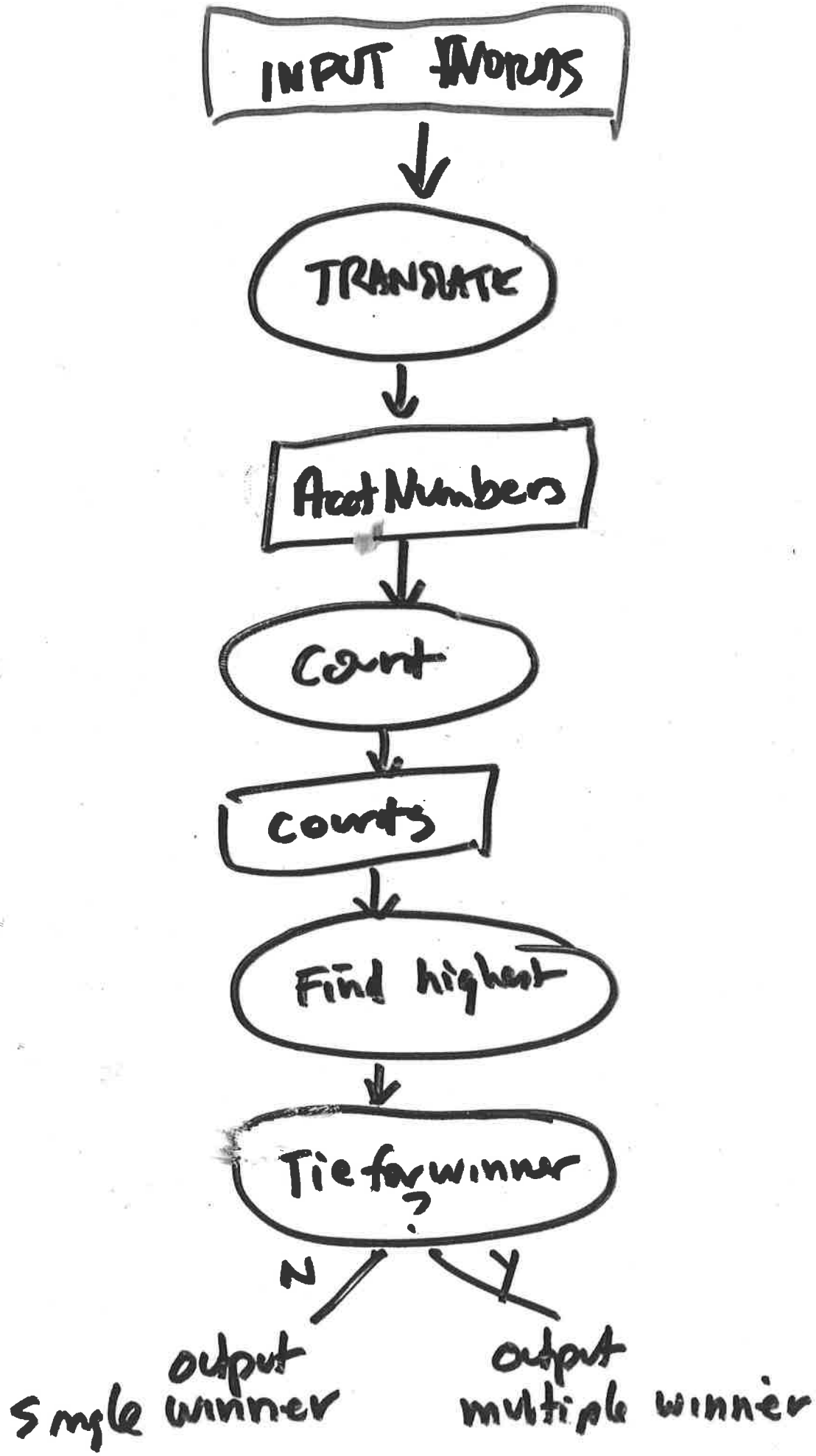
Find the highest # of occurrences.
(Iterate over map keys, get list size)

How to find words that correspond to the highest occuring number?

1. Once I know the highest number, iterate over the original list, translate each word again but check if the translation equals the highest: if yes, print.

2. Save each word along w. the translation
   Or
   Save each translation and all its words.

Map


INPUT WORDS
TRANSLATE
Acct Numbers
Count
counts
Find highest
Tie for winner
?
N
Y
output single winner
output multiple winner

Data Structure Design

After translating a word use the number as the key and add the word to the list.

map

269	boy · cow · bow
315	age - had - bad
18	go

Once the map is built finding the single highest requires iterating over the keys, get list.size() and save largest one.

How to handle ties for winner?

Approach 1

Iterate over the map as for a single winner, but create a record for each entry containing number and count, add to a list, ~~when~~ then sort the list with Collections.sort(). The top winners will be at the end of the list.

Approach 2

Have the ~~key~~ value field in the original map be modified to contain a running count (= to list.size()).
After map is built, obtain the values (a set)
and use Collections.sort() with count as sort field.

269	boy.cow.bow	3

Approach 3

Use a TreeMap which can be ordered by the key.
Modify the key to include the count. Count increments for each word added to the list. When map is complete ~~iterate over it and~~ extract top items.

263	3	boy.cow.bow

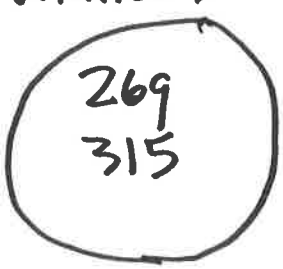
Approach 4

Iterate over map as for single winner, but maintain a running "High Count" and a "WinnersCirde" (a set of winning numbers). Tied winners are added to the set. If a number exceeds HighCount, the set is cleared and the new winner entered.

Each item in Winner Circle can be looked up in map to find its associated words.

HighCount
3

Winner's Circle



Analysis

1. requires an extra sort, expensive
2. " " " " "
3. upon reflection, won't work because updating count would alter the value of the key.
4. requires an extra data structure, Winner's circle but avoids sort.

Thinking further about 4, I realize it doesn't have to be done as a subsequent iteration after the map is built: High Count and Winner's circle can be maintained during creation of the map.

which would be faster.

Lastly, theres no reason the Values in the Map need to be a list. Order doesn't matter so use a Set.

Decision

acct words

acctMap = HashMap<String, Set<String>>;

winnerCircle = HashSet<String>

acct