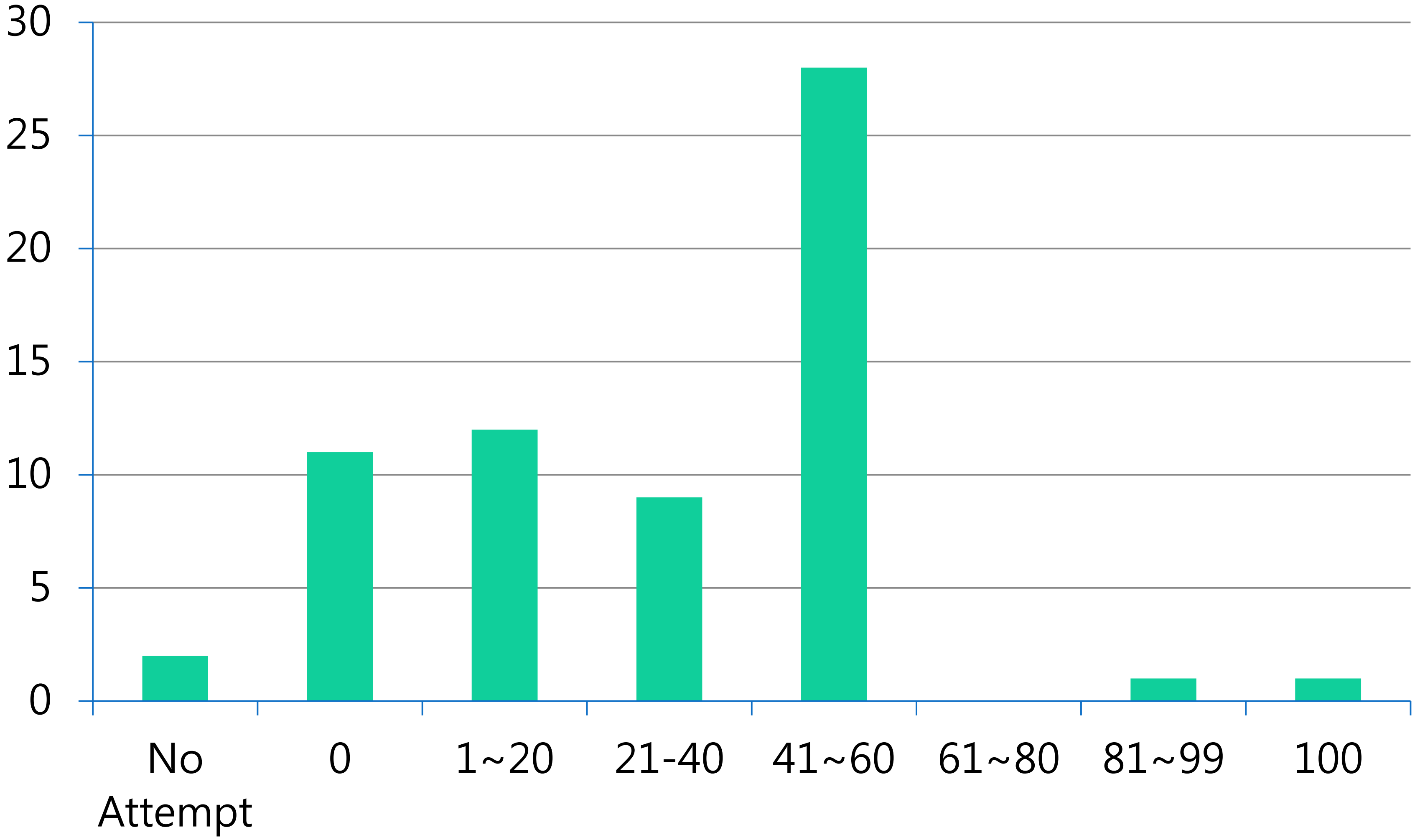
Input Method

Tony Wong
January 11, 2014

Statistics



Description

- Given a dictionary of candidate words and Dr. Jones' input, correct the words.
- For each word in the sentence, Dr. Jones mistypes exactly 1 letter.

- Size of dictionary = N
- Number of words in sentence = W
- Length of each word = L

Sample

```
6 8
computer
hardware
installs
keyboard
software
speakers
4
conputer         computer
installa         installs
compuyer         computer
softwate         software
```

Note

- In the sample below, we are using actual English words. In the test cases, however, random generated synthetic words will be used. Nearly all the words and letters are generated independently and uniformly at random (the chance of each alphabet appearing is equal). The positions of the incorrect letter are also chosen independently and uniformly at random.

- The words and letters are random.
- The position of the incorrect letter is also random.
- The dictionary is given in sorted order.

Constraints

- In test cases worth 50% of the total points
  - $1 \leq N, W \leq 2000$, $6 \leq L \leq 20$
- In all test cases
  - $1 \leq N, W \leq 20000$, $6 \leq L \leq 50$

50% Solution

- Do what it says on the question paper.
- For each word in sentence, check every word in dictionary to find a match.
- O(NWL) / O(NW)

Solution

- O(NWL)
  - 50%: 2000 x 2000 x 100 = 4e8
  - TLE, ~25%
- O(NW)
  - 50%: 2000 x 2000 = 4e6
- 100%
  - Requires more efficient algorithm
  - Time / memory trade off necessary?

Binary search

- In attempt to reduce complexity to O(W lg N)
- Recursively locate the "closest" word.

- Does it really work?
- Example:

Dictionary = {55555, 55655, 56555, 56455}

Search = 57655

- Binary search DOES NOT work.

Overly complex solution

- Hashing
- A hash is a short signature of a large data.
  - e.g. sha512(putty.exe) = 6def636ab478a7f49127c706272ff8b2862a5de50fd34e1e8509b7c1ff1da6c87001a764b5c9bb2d56d30534cfaee10c8c343575e79ae853e42c3306561411cf

Hashing Method 1

- Hash all the words in the dictionary with different positions.
  - e.g. abcdef $\rightarrow$ bcdef, acdef, abdef, abcef, abcdf, abcde
- And then calculate the hash of these 6 letters, followed by binary search
- Complexity
  - Generation: O(NL)
  - Query: O(WL lg(NL)) $\rightarrow$ 20000 x 100 x 20 = 4e7
  - Memory: O(NL)
- Does it work?
  - Special handling required

Hashing Method 2

- Hash all the words in the dictionary with different positions and changed letter.
  - e.g. abcdef -> bbcdef, cbcdef, …, zbcdef,
    aacdef, accdef, …, …, …, abcdez
- Again, perform binary search on each query
- Complexity:
  - Generation: 26 x N x L = 5e7
  - Query: O(W lg (26NL)) = 20000 x 26 = 5e5
  - Memory: O(26 x N x L) = 5e7
- Does it work?
  - Not recommended

"Ideal" solution (Tony's solution)

- We have no intention to test hashing
- There is exactly one incorrect letter.
- If the first letter is different, the second one must be the same in order to be a candidate.
  - Only search those with same first letter OR same second letter.
- We know how to do same first letter, but what data structure for same second letter?
  - Create 2D array of size 26 x N to store the indexes

"Ideal" solution (Tony's solution)

- Complexity:
  - Generation: O(N)
  - Query: NW/26 = 1.5e7
  - Memory: 26 x N

- May work, or may not work

Further improvement

- Check even small part of dictionary for each query
  - Only search those with
    (same 1st letter AND same 2nd letter) OR
    (same 3rd letter AND same 4th letter)
  - What data structure to use?
  - Memory = 2 x 26 x 26 x N?

- Linked lists
  - Create two "indexes" (each size = 26x26) that contains the heads of the lists, e.g. aa, ab, ac, ad
  - Each word in dictionary will point to the next one with same 1st & 2nd letter; and same 3rd & 4th letter

Complexity

- Generation: O(N)
- Query: O(NW/26/26) = 600000
- Memory: O(N + 26x26x2)

- Since minimum L = 6, we can even do 3 letters.

Other solutions

- Hash table
- Tree

Takeaways

- Read problem statement carefully
- Random data is everywhere
- Don't overcomplicate things
- Full solution may be leaner than you thought

- Further reading:
  - Linked lists
  - Database string indexing
  - Hashing