Find the most common character of a string.

Potential questions from the candidate:

- What is the expected length? Typically between 30 and 2000 characters.
- What is the encoding of the characters? Answer: ASCII characters, 7 bits each.
- Some candidates might reasonably think you mean “letter” rather than character. If so, kindly correct them.

To keep track of the counts, candidates typically use either:

- an array of 2^7 = 128 integers with the character value used as an index
- a hash table
- a map

Ask what the algorithmic complexity would be, if the candidate doesn’t say. If the candidate specifies a tree-based map, which is O(N log N) complexity, as if there’s a faster way.

If the candidate hasn’t written a function to do this yet, ask that she/he do so.

Ask the candidate to suggest some unit tests for the code. Does the candidate consider the edge case of an empty string?

Ask how the approach would change if the count function is called repeatedly on short strings. Say, fixed at 8 character length. If the candidate is using an array, see if the candidate notices that zeroing it gets relatively expensive.

What if the encoding is unicode, with 16 bits per character, for 2^16 = 65536 possibilities. How might that change the approach?

What if the string is extremely long? So long that it can’t fit on a single machine?

What if sending the results between machines is expensive? How could the network traffic be reduced?

BLUE

Design a system to calculate the “top 10 most popular Google queries” between any two dates.

This is a system design question. It is intentionally very open-ended. The idea here is not to reach a “correct” solution, but to get an idea of the candidate’s way of thinking and general knowledge of large-scale design and its tradeoffs.

Potential questions from the candidate:

- What do you mean by “most popular”?
- What kind of queries? (Most people will assume text search, but there are other options.)

There are a lot of directions the candidate can go with this:

- Batch reports? Realtime console?
- Arbitrary dates vs specific intervals (eg, yearly report)? Date granularity?
- Allowance for filters (e.g., limit to U.S. IPs)?
- Canonicalization (e.g., correction of typos)?
- Mobile app? Web app? Desktop app?

Hopefully the candidate identifies some components of the system. If not, prompt her/him to do so. Ask limited questions at first (e.g., “How would you handle data collection?”) and then ask more specific questions if the candidate struggles.

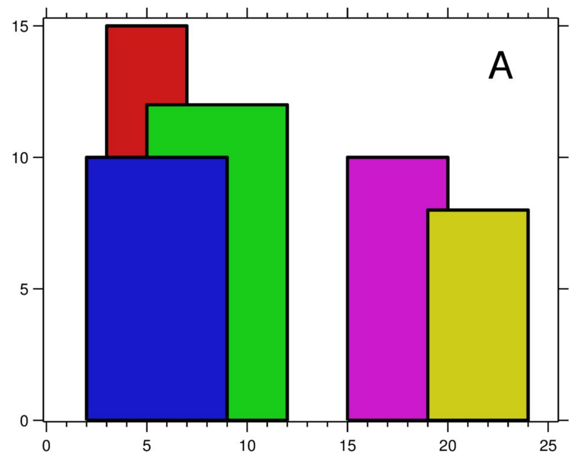
- Collection: How do you collect search queries? When to canonicalize?
- Storage: How do you store them, and for how long? How much space? What kind of summary can be stored, and how granular?
- Processing: How would the pipeline be designed? Are results done offline (i.e., precomputed) or in real time in response to requests? Can results be cached?
- Frontend Operations: Add/Delete/Save a report? Settings management?

Ask the candidate for some back-of-the-envelope analysis of how much data would need to be processed, and what resources it would require. This is basically a Fermi problem in which the candidate might need to estimate the rate of search queries sent to Google (order of 10 billion per day), the average length of a query string, and the size of metadata that accompanies it.

RED

[DESIGN QUESTION]

Compute the silhouette edge of a skyline.



The skyline is made up of rectangular buildings. Each building goes from L to R along the x-axis and from 0 to H along the y-axis. The question is how to compute the points that define the silhouette edge.

Potential questions from the candidate:

- Are the left / right / height values integers? Initially say yes. If you want to make the problem harder later, you can later make them floating point.
- Are the buildings sorted in any way? No.
- How should the skyline be represented? This is up to the candidate. A good option is a list of X,Y points at the skyline vertices. Another option (less efficient) would be a height for every X position. If they go this way, that's initially OK, but later ask that they switch to float coordinates.

The candidate might come up with a brute-force solution; for instance, an O(N * M) solution that iterates through N points on the x axis and at each one loops through all M buildings. That's fine to start, but ask if it can be done better.

A better solution is to iterate over all the x values associated with building edges. That's O(M^2), but M is likely smaller than N.

You can get it to O(M log M) — the complexity of sorting the building edges — with a properly structured algorithm. This will probably process edges from left to right, maintaining a list of the "active" buildings as it goes along.

Does the candidate's code work for edge cases? For instance, what if the list of buildings is empty? It should return a zero-height skyline. Does it?

If the candidate does not talk about algorithmic complexity, ask. Approximately how much memory does the algorithm use?

If time allows, ask the candidate to suggest some unit tests for the code.

YELLOW

Determine an ordered alphabet from a dictionary of words.

Given a dictionary (a list of words in sorted, alphabetical order) of words in an unknown language, find the alphabet (an ordered list of letters) of that language.

Example dictionary:
art
rat
cat
car

Alphabet is: [a, t, r, c]

A good candidate will go over the word list and store the "*x* precedes *y*" relationships of letters in a graph. The candidate would then sort the relationships. (This kind of sort is called a "topological sort": rather than ordering a list, it orders a graph.)

Some hints to give if the candidate gets stuck:

- "Dictionary" is a somewhat loaded word, especially if the candidate uses Python. If the candidate seems confused on this, I'd immediately clarify.
- If we know the order just between two words, what information about the letter order can we extract? For instance, if we know that "art" is before "car", then we know that *a* is before *c*. If we know that "car" is before "cat", we know that *r* is before *t*.
- Ordering is transitive. We don't need to check each pair of words.
- I wouldn't expect candidates to be familiar with topological sort, so I'd be willing to help a good bit here. Some leading questions include: What are potential candidates for the 1st letter in the alphabet and why? Can we compute the alphabet incrementally?

Does the candidate consider edge cases?

- Empty word list?
- Small word list? The candidate might realize that if the dictionary results in a disjoint letter graph, then there will be multiple alphabets that can work.
- Upper- vs. lower-case letters? Consider them the same. *X* matches *x*.
- Numbers or symbols in the "words"? Great question, but let's just stick to the 26 letters.

What is the algorithmic and memory efficiency of the candidate's approach? Would it fit in memory for the entire English dictionary? (There are roughly 200,000 words in the Oxford English Dictionary, but hopefully the candidate will come up with a reasonable estimate here. In computing terms, it's still tiny.) What about an alien language with billions of words?

GREEN