**Coding Competency: Data Structures & Algorithms**

**What is Data Structures & Algorithms?**

This coding session is all about diving deep on fundamentals. It’s about understanding the usage of knowing algorithms and knowing data structures in a unique way. With the knowledge of basic algorithms between SDE I, II, and III, there’s not many differences. However, we do expect between the levels of SDEs to come up with innovative ideas.

The round is not a game to verify “Hey, let’s verify what you studied in Leetcode and let’s see if you know it or not.” Amazon interviewers don’t care about what you studied in Leetcode. This is not our process so please don’t make this assumption when you go into these coding rounds.

We focus on hash tables, sets, stacks, arrays, lists. Interviewers are not okay with surface level answers. If you decide to use a hash table and mention this to the interviewer, the interviewer is going to ask why you chose this. If you just give one word answers that are surface level without justifying your explanation, you will lower the bar. Interviewers will challenge you and here are some follow up questions they may ask:

* *Can you think about other data structures?*
* *What about collisions? Explain what type of algorithm would be a good choice to handle collisions?*
* *Why did you choose to use this data structure or algorithm?*

SDE I – Select data structure, explain why you selected the data structure, and use it even though it may not be an optimal one.

SDE II & III – Select data structure, we want candidates striving to find the optimal data structure. We want to see candidates speaking about the tradeoffs. For example, talk about the advantages of using an array or a list. Tell the interviewer, “Based on the question you’re asking me interviewer, if I use an array, here is the tradeoff of using it versus using a list.” They want to see you use multiple data structures.

Prior to coding, we see bar raising candidates that spend 12 minutes with the interviewer having a discussion before they go to the code in this round. Once you asked clarifying questions, removed the ambiguity, you can now start coding which should be completed within 18 minutes. In total, this coding session should be completed in 30 minutes.

**How to Pass Data Structures & Algorithms?**

\*In the chart below, there is no absolute scoring on the criteria and they are equivalently important. It is about making decisions based on trade-offs and risks seen in the interview (and obviously the more mild strength/full strength data points the better).

|  |  |  |
| --- | --- | --- |
| **Criteria** | **Mild Strength** | **Strength** |
| **Code** | Able to execute the implementation of what they described. A few minor problems but no big conceptual errors. | Fully complete, tidy. Elegantly solves problem. May even write tests. |
| **Variety of known structures**  Linked lists/arrays/hash tables/trees/tries/graphs/queues/  stacks/deques/vectors/skip lists/  linked hash maps/heaps | Knows and understands many ways to organize data. | Knows and understands more sophisticated or composite types. |
| **Understand ramifications of changing structures**  (code/impact/performance) | Understands performance and functional impact of a data structure swap. Understands basic trade offs. | Perceives and can adapt to more subtle ramifications, like impact on space complexity or data locality. Comprehensive understanding of trade offs. |
| **How to use a structure** | Picks an appropriate structure. Able to work with it in code to get the job done. Knows the basic operations (specifics aren't required). | Easily applies the chosen structure with no guidance. Can easily shift among multiple structures. Knows more advanced operations. |
| **Know how structures work internally** | Can explain for multiple structures how the way in which they are organized leads to the performance of their methods. For instance, that a hash table uses a hash code and the ability to O(1) jump in an array to have an O(1) get, or that a [balanced] BST uses left-right ordering and binary search to have a O(log n) get, etc. These specific examples are not required. | Full and detailed knowledge of the inner workings of several non-trivial structures. For instance, explaining multiple ways in which hash tables manage collisions and how that leads to amortized but not true O(1) performance, or of how to percolate values up or down a heap, etc. These specific examples are not required. |
| **Edge Cases** | Able to identify edge or special cases. Able to properly handle edge and error cases (conditions/exception handling/logging) gracefully and without causing ambiguous response (eg. returning -1 where -1 is a valid response).  Considers happy case as well as at least one edge or error case. | Thorough understanding of potential inputs and how they would be handled in the code. Edge and error cases are handled gracefully and do not create ambiguous results. Thorough brainstorming on test cases, may write out tests, confirms they work in the given code. Where a case is not handled they should be able to discuss proper handling (not necessary to update the code for all). |
| **Performance of Algorithm** | Understands Big O time and space complexity. Able to identify the correct complexity. Able to discuss multiple solutions and discuss trade offs in time and space complexity. | Understands and correctly identifies Big O time and space complexity. May discuss amortized complexity.  Able to discuss trade offs and practical situations where each would be used.  Able to give additional impacts of increase in complexity (costs). |
| **Compare Solutions** | Able to describe at least two solutions and compare and evaluate the data structures and algorithms required for each.  Includes a discussion on complexity. Able to firmly choose a solution and justify the choice. | Proactively identifies and compares two or more solutions. Good depth in discussion of trade offs and understanding of the data structure and algorithm. Various solutions include a high variety of data structures. |

**Sample Questions & Solutions**

***Question #1: Computer Game***

You're chief gameplay scientist for the next hottest web start-up. This is an online word game where people can play against a computer. The game has a main goal to build a word using letter. Higher score = best word.  
You need to provide a proof of concept function which will choose the best word to play, given a set of letter tiles, a table of letter scores, and a dictionary of valid words.

**NOTES for interviewer:**

* There are multiple questions the candidate should ask to clear ambiguity.
* This question mirrors a scrabble game but knowing the game Scrabble is NOT needed to be successful to solve this problem and does not put a candidate at a disadvantage or advantage based off the level of Scrabble knowledge they may have.
* Python/JavaScript - Tri structures (node class)/graphs are used in JavaScript which is the optimal solution to solving this problem. Can use brute force as well.
* Go - Tri structures/graphs are used in GO which is the optimal solution to solving this problem. Can use brute force as well.

**Common Questions asked by candidates you should be prepared to answer as the interviewer:**

1. Is capitalization important? (no)
2. What’s the character set? (A-Z)
3. Do we have to handle blank tiles? (Not initially, but this is a great follow-up question if you have time.)
4. How many letter tiles do we have (normally, 7, but we might want to handle more or less than this as well).
5. Can there be duplicate letters (yes)
6. What are the expected input and output formats? (see below)
7. What are the expected data sizes? (average word length, number of inputs).
8. What if is more than one ‘best’ move (two words have the same highest score)? (you can pick any one)

**Follow up questions as the interviewer for L4 SDE data points:**

* Brute force solution is expected - probe to get them to this.

**Follow up questions as the interviewer for L5 SDE data points:**

* To assess bar raising L5 SDE data points: A) How would you handle blank tiles? They are worth no points but can be used in place of any letter. B) Another question is to ask for the top-N best words, rather than only the best one. \*\*this could be a simple change, but also could add a lot of complexity\* C) Implement the Tri.

**Follow up questions as the interviewer for L6 SDE data points:**

* All SDE II follow up questions are answered - similar coding bar maybe mindset around scaling? Long term thinking? Etc?

***Question #2: Collection of Strings***

Given a collection of strings, write a function that will return the strings grouped into groups where all words in a group are anagrams of each other.

Example:  
  Input: opt, bin, foo, top, cat, nib, act, pot  
  Output:  
        opt, top, pot  
        bin, nib  
        foo  
        cat, act"}}}

**NOTES for interviewer:**

* **For this question, the example is needed to give the candidate to outline the problem statement but there is still a level of ambiguity to the question and follow up questions the candidate should use. Anagram definition does not come up frequently from the candidate but interview should be ready to define this for the candidate.**
* No call outs needed for JavaScript or Go around priority queues to solve this problem.
* Focused as an algorithmic question more than a data structure question
* Clarifying Questions
  + Is capitalization important? Is whitespace important? What’s the character set?
  + What are the expected input and output formats? (ie. is input an array, vector, list... how should output be returned? (collection of collections)
  + What are the expected data sizes? (avg. word length, number of inputs).
  + How should duplicate inputs be handled?
  + Does the order of the output matter?
  + Outline what DS and algorithms can be used to solve it?

**Follow up questions as the interviewer for L4 SDE data points:**

* SDE I would be that they got the brute force solution, and cannot think of other ways to solve it, so interviewer could ask them to talk through time & space complexity and probe towards one specific solution and the expectation for bar raising data for SDE I is they can take the hints and probes and work towards a solution at that point once given the hints.

**Follow up questions as the interviewer for L5 SDE data points:**

* A) How they would change their solution if it was going to be applied to DNA sequences, instead of common English words (DNA sequences are strings with millions of characters, but using only the characters A,C,G,T).
* B) What would you do if you had too many words to fit into memory? Not a code change but good talking point to walk through removing variance and duplication (maybe through using map reduce although more options to handle this other than map reduce). This would rely on an interviewer to be exposed the map reduce which not everyone is. Would not ding candidate or have it be a concerning indicator if the candidate could not talk about this either. It is a nice to have question to ask if candidate is doing very well and additional talking points could be of value.

**Follow up questions as the interviewer for L6 SDE data points:**

* If candidate can talk through A and B as coding bar is similar and aligned from SDE II to SDE III. A) How they would change their solution if it was going to be applied to DNA sequences, instead of common English words (DNA sequences are strings with millions of characters, but using only the characters A,C,G,T).

***Question #3: Next Greater Element***

Given an array, print the Next Greater Element (NGE) for every element. The Next greater Element for an element x is the first greater element on the right side of x in array. Elements for which no greater element exist, consider next greater element as -1.  
Examples:  
  
For any array, rightmost element always has next greater element as -1.  
For an array which is sorted in decreasing order, all elements have next greater element as -1.  
For the input array [4, 5, 2, 25], the next greater elements for each element are as follows:  
Element NGE  
4 → 5  
5 → 25  
2 → 25  
25 → -1

**NOTES for interviewer:**

* The goal is to figure out how to traverse an array to find next highest value or just goal is to find clever use of array or loop.

**Follow up questions as the interviewer for L4 SDE data points:**

* Get through brute force, interview gives them probes and hints towards a solution and candidate can take it from there.

**Follow up questions as the interviewer for L5 SDE data points:**

* Talk through time and space complexity. Can talk through and/or implement one of the optimal solutions. Stack approach comes up most often.

**Follow up questions as the interviewer for L6 SDE data points:**

* N/A

***Question #4: Baby Blocks***

You have a collection of 'baby blocks' (cubes with single upper-case letters of the alphabet on each side). Each block could have up to six different letters on it, assume that each block only has two distinct letters.  
  
Write a function that takes a collection of blocks and a target word and returns true or false depending on whether or not you can spell the target word with the collection of blocks.  
  
Ex: (B,A),(A,B),(X,Y),(A,B): "BABY" => yes (B,A),(A,B),(L,E),(A,B): "ABLE" => no (since L and E are on the same block).

**NOTES for interviewer:**

**Follow up questions as the interviewer for L4 SDE data points:**

* No follow up questions, just get through base case solution with optimal solution (hinting or some probing OK)

**Follow up questions as the interviewer for L5 SDE data points:**

* Follow up questions to ask an SDE II: 1) What changes if you have 6 different letters on a cube? 2) How would you change the approach to handle a blank side that can be any letter on the cube? - we expect an SDE II bar raising candidate to get through one of these questions, maybe talk through the other if there is time. candidate will not be able to usually get farther in the time constraints. Always ask time and space complexity.

**Follow up questions as the interviewer for L6 SDE data points:**

* Same as SDE II questions apply, get through at least one of the follow up questions, candidate will not be able to usually get farther in the time constraints

***Question #5: Compound Words***

You are given a (potentially large) list of words.

Some of these words are compound (combination) words, where all word parts are also in the list.

Example list:

[rockstar, rock, star, rocks, tar, star, rockstars, super, highway, high, way, superhighway]//

The task is to identify all compound words, where all parts of the words are also in the list.

The output would be a list of lists.

Example compound word combinations:

rockstar → rock + star

superhighway → super + highway

superhighway → super + high + way//

Example output list of lists

[[rock, star], [super, highway], [super, high, way],...]

**NOTES for interviewer:**

* There can be many nuances to accurately solving this problem. Expected candidate will ask some follow up questions to disambiguate. When SDE's have tested it and tried to solve it, it is hard to solve, may be a lot of code to make it work. Help simplify this question for the candidate.

**Follow up questions as the interviewer for L4 SDE data points:**

* Get through brute force, interview gives them probes and hints towards a solution and candidate can take it from there.

**Follow up questions as the interviewer for L5 SDE data points:**

* Talk through time and space complexity. Can talk through and/or implement one of the optimal solutions. Stack approach comes up most often.

**Follow up questions as the interviewer for L6 SDE data points:**

* N/A

**Data Structure Reference**

**Array / Linked list / Queue / Stack / Vector**

An ordered collection of values. These are very basic structures that are ubiquitous in software and is a built-in data type in some form for almost all languages. Because of the ubiquity, candidates should know the space and time trade-offs of various forms.

**Hashmap**

Key / value pairs that use a hashing of the key to map to the value. This data structure is ubiquitous in practical usage and is a built-in data type for several languages. There are many variations to suit specialized needs. Can be paired with a list to impart ordering, e.g. LinkedHashMap. Because of the ubiquity, candidates should know the space and time trade-offs of various forms.

**Set**

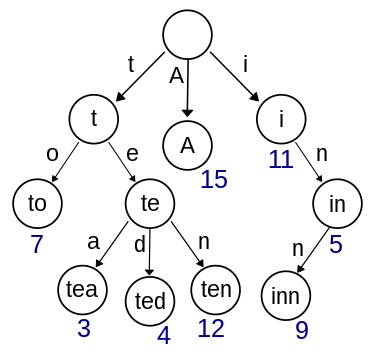
A unique collection of unordered values. Many implementations are built on top of a HashMap. Can be paired with a list to impart ordering, e.g. LinkedHashSet. Because of the ubiquity, candidates should know the space and time trade-offs of various forms.

**Tree / Binary Search Tree**

A hierarchical structure consisting of a single root node and one or more sub-trees as child nodes where any child node can have one and only one parent. A binary tree adds the constraint that any node can have at most two children. A binary search tree adds the constraint that the “left” child is of a lesser value than the “right” child. A tree is a highly constrained form of graph. While direct usage of trees in programming is limited, the concept is simple and has representations in every day computing, e.g. file systems. There are many variations to suit specialized needs.

**Trie**

A specialized form of tree that is used to store a dynamic set or associative array where keys are usually strings. No node in the tree stores the key associate with the node; instead its position in the tree defines the key which it is associated.



**Graph / Directed Graph**

A finite set of nodes that are connected by one or more edges. Edges between nodes may be directional (Directed Graph), i.e. a node can be traversed to only if the edge is in the direction of that node. Nodes generally hold one or more values. Edges can be imbued with properties to impart meaning to the connection, e.g. weights or labels. Direct usage of graphs is limited but growing, and the concept is easy to explain.

**Heap / Priority queue**

A specialized form of tree where the nodes are ordered according to their value. In a max heap, the parent node is always >= the value of all of its children. In a min heap, the parent node is always <= all of its children. While not all priority queues are implemented using a heap, a heap is the most common method.

This data structure is questionable to use because:

* 1. Questions that use heaps as an optimal data structure tend to fall into a specific type: Get me the highest/lowest X pieces of data from a set. These questions are typically inflexible on the data structure selection.
  2. Heaps are not commonly used in day-to-day development, so candidates either know it or they don't. It is difficult to guide a candidate into a heap structure if they have no knowledge of it.

**Directed Acyclic Graph (DAG)**

A specialized form of graph where edges must be directed and the graph is free of cycles, i.e. any traversal of the graph will not visit the same node more than once. The commit structure of the popular Git source control system is a DAG

This data structure is questionable to use because:

* 1. Questions of this type are typically heavy on algorithm choice, but light on data structure choice. Constructing a question that contains both typically makes for a question too complex to ask at a whiteboard interview

**Bloom Filter**

A probabilistic, strictly additive, data structure that provides a space efficient way of answering the question if a value is “probably in the set” or “definitely not in the set.” It consists of a bit array and one or more hashing functions. On add, the hashing functions are applied to the value and the corresponding bit in the bit array is flipped to 1.

This data structure is questionable to use because:

1. Bloom Filters are not used in traditional software development. A candidate answering this type of question would require previous specialized knowledge of their use.

**Red-Black Tree/ B-Tree / Self Balancing Tree**

A specialized form of binary tree that performs balancing on insertion. Details about how trees self-balance are arcane and complex enough that existing knowledge should not be assumed. It should also not be assumed that the details of how the algorithms work can be taught during an interview in a way that does not detract from the interview. Usage that solely relies upon the space and time complexity of search, insert, and delete could be considered fair if those complexities are given to the candidate.

**Union**

More of a data type than a data structure, a Union can hold a single value where that single value can have one of the list of types that define the Union. Unions are primarily present in languages that provide low-level access to memory manipulation, e.g. C/C++.