105 Data Structures interview questions to ask recruiters

Questions

1. Can you describe what an array is like explaining it to a child?

2. Imagine you're organizing your toys. How would you use a linked list to keep track of them?

3. What is a stack, like a stack of pancakes or stack of plates?

4. Queues are like waiting in line. How does that work in data structures?

5. Why would you choose an array over a linked list, and vice versa?

6. What's the difference between a stack and a queue in simple terms?

7. How do you add something to the end of an array when it is already full?

8. Can you explain the concept of searching within an array?

9. What is sorting and why would we want to sort elements in an array?

10. Explain what a hash table is and how it helps us find things quickly.

11. How does a binary search work, and why is it faster than a regular search?

12. What are the advantages and disadvantages of using linked lists?

13. How would you implement a queue using two stacks?

14. Explain the concept of a 'Last-In, First-Out' (LIFO) data structure.

15. Explain the concept of a 'First-In, First-Out' (FIFO) data structure.

16. How do you prevent stack overflow and underflow?

17. How are data structures stored in memory?

18. What is the difference between a linear and a non-linear data structure?

19. How can you improve the efficiency of searching in a data structure?

20. What are some common applications of queues in real-world scenarios?

21. How does recursion relate to the use of stacks in data structures?

22. What does Big O notation mean and how does it relate to the efficiency of data structures?

23. When would you use a hash table over other data structures for searching?

24. Explain how you would reverse a linked list.

25. Describe how you would detect a loop in a linked list.

26. How do you deal with collisions in hash tables?

27. How do you efficiently find the median in a stream of unsorted numbers?

28. Explain how a Bloom filter works and what its use cases are. What are its limitations?

29. Describe how you would implement a Least Recently Used (LRU) cache.

30. How do you detect a cycle in a directed graph?

31. Explain the difference between a B-tree and a binary search tree. When would you use one over the other?

32. How would you implement a trie data structure and what operations would it support?

33. Describe the difference between a disjoint set and a union-find data structure, and how can you implement each of these?

34. How do you find the shortest path between two nodes in a weighted graph?

35. Explain how a skip list works. What are the advantages of using Skip list over other data structures.

36. How do you efficiently implement a stack using only queues?

37. Describe the concept of self-balancing trees. Explain the purpose and mechanism.

38. What are the different ways to implement a priority queue? What are the pros and cons of each approach?

39. How would you design a data structure to support range queries (e.g., find the sum of elements within a given range) efficiently?

40. How can you serialize and deserialize a binary tree?

41. Explain the concept of a segment tree and its applications.

42. Describe how you would implement a min-max heap.

43. Given a large dataset that doesn't fit in memory, how would you sort it?

44. How do you implement a concurrent hash map that can handle multiple threads safely?

45. Explain the difference between Breadth-First Search (BFS) and Depth-First Search (DFS) and when would you prefer one over the other?

46. How would you implement a data structure to store and efficiently retrieve data based on geographical coordinates?

47. Explain the concept of a Fenwick tree (Binary Indexed Tree) and its applications.

48. Describe how you would implement a data structure that supports efficiently finding the k-th smallest element in a stream of numbers.

49. How do you use dynamic programming to solve problems related to trees or graphs?

50. Explain what an AVL tree is and the rotation operations required to maintain its balance.

51. How would you efficiently find all anagrams in a large list of words?

52. Explain the trade-offs between different implementations of B-trees (e.g., B-tree, B+ tree, B^+ tree).

53. How can you implement a persistent data structure and what are the benefits?

54. Describe the use cases for a disjoint-set data structure (Union-Find) and how it optimizes connectivity problems.

55. Explain how to implement an efficient auto-completion system using Tries.

56. How does a Bloom filter work, and what are its applications and limitations in probabilistic data structures?

57. Describe the advantages and disadvantages of using a skip list compared to other ordered data structures like balanced trees.

58. Explain how to use a segment tree or Fenwick tree to efficiently solve range query problems.

59. How can you design a data structure to efficiently store and query spatial data (e.g., using a quadtree or KD-tree)?

60. Describe the concept of a self-organizing list and how it can improve performance in certain scenarios.

61. Explain how to implement a data structure that supports both efficient insertion and retrieval of elements based on priority (e.g., a pairing heap or Fibonacci heap).

62. How do you handle concurrency issues when multiple threads access and modify a shared data structure?

63. Describe the differences between various graph traversal algorithms and their applications

64. Explain different collision resolution techniques in hash tables and their impact on performance.

65. How do you design an in-memory cache using different eviction policies (LRU, LFU, FIFO)?

66. Describe the concept of a Patricia trie and how it optimizes space usage compared to a standard trie.

67. Explain the use of a Minimax tree in game AI and decision-making problems.

68. How can you use a suffix tree to efficiently solve string-related problems such as finding repeated substrings?

69. Describe the implementation and use cases of a space-efficient probabilistic data structure like a Count-min sketch.

70. Explain how to build and use a Voronoi diagram to solve proximity-based problems.

71. How can you efficiently find the median of a large stream of numbers using a combination of data structures?

72. Describe the structure and usage of a BSP tree in computer graphics for spatial partitioning.

73. Explain how to implement a distributed hash table (DHT) for storing and retrieving data across a network.

74. How would you design a data structure to efficiently track the frequency of events over time?

75. Describe the application of a range tree in solving windowing queries or finding all points within a given range.

76. How can you design a data structure that efficiently supports both finding the median and inserting new elements?

77. Describe how you would implement a persistent data structure using linked lists or trees, focusing on memory efficiency and immutability.

78. Explain the trade-offs between using a Bloom filter versus a standard hash table for membership testing in a large dataset.

79. How would you go about designing a data structure to track the frequency of words in a very large text corpus in real-time, subject to memory constraints?

80. Can you elaborate on how you might optimize a trie data structure to minimize memory usage while still providing efficient prefix searching?

81. Describe a scenario where using a skip list would be more advantageous than using a balanced tree, and explain why.

82. How would you implement a data structure that supports efficient range queries (finding all elements within a given range) in a multi-dimensional space?

83. Explain how you can use a disjoint-set data structure to solve network connectivity problems, and what optimizations can be applied.

84. Describe how you would implement a data structure that supports efficient insertion, deletion, and search operations while also maintaining the elements in sorted order.

85. How can you design a data structure that automatically expires old data based on a least recently used (LRU) policy?

86. Discuss the advantages and disadvantages of using a B-tree versus a B+tree in database indexing.

87. Explain how you would implement a concurrent queue data structure that supports multiple producers and consumers without race conditions.

88. Describe how you could use a Fenwick tree (Binary Indexed Tree) to efficiently compute prefix sums in an array.

89. How would you design a data structure that allows you to efficiently find the k-th smallest element in a stream of numbers?

90. Explain how to use a suffix tree to solve complex string matching problems, like finding the longest common substring of multiple strings.

91. Describe a scenario where a self-balancing binary search tree (like an AVL tree or Red-Black tree) would be essential for good performance.

92. How would you implement a data structure to efficiently store and retrieve geographical data (e.g., points on a map)?

93. Explain how you would design a data structure for implementing an auto-complete feature, optimizing for both speed and memory usage.

94. Discuss how to use a graph data structure to model social networks and find influential users.

95. How would you implement a data structure that supports efficient union and find operations on sets of elements, with path compression and union by rank?

96. Explain how you would go about creating a copy-on-write array data structure.

97. Describe how you would implement an immutable stack data structure.

98. How do you handle collisions in a hash table and what are the impacts of different collision resolution techniques on performance?

99. Explain how you can adapt a standard data structure like a hash table to handle concurrent access from multiple threads, ensuring thread safety.

100. Describe the steps for implementing a Bloom filter and explain its usage in applications like caching and anomaly detection.

101. How can you use a segment tree data structure to efficiently solve range query problems, like finding the minimum or maximum value in a range?

102. Explain the benefits and drawbacks of using a sparse matrix data structure versus a dense matrix, particularly in the context of large datasets.

103. How would you implement a data structure to manage a large number of time-series data, supporting efficient querying and analysis?