When interviewing for a dead code detection project, consider asking the following questions:

What is dead code, and why is it a concern in software development?

How can you identify dead code in a codebase?

What are the potential risks of removing dead code automatically?

Explain the difference between compile-time and runtime dead code.

How can you prevent the introduction of dead code in a project?

What strategies can be employed to eliminate dead code in a legacy system?

How does dead code impact performance, and why should it be addressed?

Can dead code be beneficial in certain scenarios?

Discuss the role of code coverage in dead code detection.

What are some challenges in eliminating dead code from large codebases?

Can you explain the algorithm or approach you would use to detect dead code?

How do you differentiate between unreachable and potentially dead code?

How does your approach adapt to different programming languages?

Are there language-specific challenges you foresee in dead code detection?

Have you worked with any existing dead code detection tools?
Which ones?

What programming languages and environments is your approach compatible with?

How do you handle false positives and false negatives in dead code detection?

Can you provide examples of scenarios where false results might occur?

How easily can your solution be integrated into existing codebases?

Is your approach scalable for large projects, and how does it handle codebases with multiple dependencies?

What are the time and space complexities of your dead code detection algorithm?

How do you optimize for performance while ensuring accuracy?

Can your solution integrate with version control systems to track changes over time?

How does it handle branches and merging in version-controlled code repositories?

How do you present the results of dead code detection to developers?

Are there visualizations or reports generated to aid developers in understanding and addressing the identified issues?

How do you plan to maintain and update the dead code detection tool as programming languages evolve?

Are there mechanisms for community contributions or customization?

How do you validate the accuracy of your dead code detection approach?

Are there testing strategies in place to ensure the reliability of the tool?

Can you explain the difference between static and dynamic dead code analysis?

How would you choose between them for a given project?

What techniques or tools have you used in the past for identifying dead code? Share an example of a successful dead code elimination experience.

How would you handle the detection and removal of dead code in a large and complex codebase?

Explain the potential impact of removing dead code on software maintainability and performance. How would you mitigate any risks associated with code elimination?

Describe a scenario where a piece of code might appear to be dead but is, in fact, still essential. How would you ensure accuracy in your dead code analysis?

What strategies would you employ to convince stakeholders and team members of the benefits of investing time and resources in dead code detection and elimination?

How do you prioritize dead code elimination alongside other development tasks, especially in a project with tight deadlines?

Discuss any challenges you anticipate when implementing dead code detection in a legacy system. How would you approach overcoming these challenges?

Can you outline a process for incorporating dead code detection into the continuous integration/continuous deployment (CI/CD) pipeline of a software project?