```
use std::collections::{HashMap, HashSet, VecDeque};
use rand::Rng;

fn adhd_find(graph: &HashMap<char, Vec<char>>, start: char, target: Option<char>) -> Option<
    let mut visited = HashSet::new();
    let mut stack = vec![start];
    let mut queue = VecDeque::new();
    queue.push_back(start);

    let mut rng = rand::thread_rng();

    while !stack.is_empty() || !queue.is_empty() {
        if rng.gen_bool(0.5) && !queue.is_empty() {
            // BFS step
            if let Some(node) = queue.pop_front() {
                if !visited.contains(&node) {
                    visited.insert(node);
                    println!("Visiting {} (BFS)", node);

                    if Some(node) == target {
                        return Some(node);
                    }

                    if let Some(neighbors) = graph.get(&node) {
                        for &neighbor in neighbors {
                            if !visited.contains(&neighbor) {
                                queue.push_back(neighbor);
                                if !stack.contains(&neighbor) {
                                    stack.push(neighbor);
                                }
                            }
                        }
                    }
                }
            }
        } else if !stack.is_empty() {
            // DFS step
            if let Some(node) = stack.pop() {
                if !visited.contains(&node) {
                    visited.insert(node);
                    println!("Visiting {} (DFS)", node);

                    if Some(node) == target {
                        return Some(node);
                    }

                    if let Some(neighbors) = graph.get(&node) {
                        for &neighbor in neighbors {
                            if !visited.contains(&neighbor) {
                                stack.push(neighbor);
                                if !queue.contains(&neighbor) {
                                    queue.push_back(neighbor);
                                }
                            }
                        }
                    }
                }
            }
        } else {
```

```
            // Fallback to BFS when stack is empty but queue is not
            if let Some(node) = queue.pop_front() {
                if !visited.contains(&node) {
                    visited.insert(node);
                    println!("Visiting {} (BFS - fallback)", node);

                    if Some(node) == target {
                        return Some(node);
                    }

                    if let Some(neighbors) = graph.get(&node) {
                        for &neighbor in neighbors {
                            if !visited.contains(&neighbor) {
                                queue.push_back(neighbor);
                                stack.push(neighbor);
                            }
                        }
                    }
                }
            }
        }
    }

    None
}

fn adhd_search(graph: &HashMap<char, Vec<char>>, start: char, target: Option<char>) -> Optic
    adhd_find(graph,start,target)
}

fn main() {
    let mut graph = HashMap::new();
    graph.insert('A', vec!['B', 'C']);
    graph.insert('B', vec!['D', 'E']);
    graph.insert('C', vec!['F']);
    graph.insert('D', vec![]);
    graph.insert('E', vec!['F']);
    graph.insert('F', vec![]);

    println!("Running ADHD search and find:");
    let result = adhd_find(&graph, 'A', Some('E'));
    match result {
        Some(node) => println!("Found target: {}", node),
        None => println!("Target not found"),
    }

    println!("0unning ADHD search and find without a specific target:");
    let result = adhd_search(&graph, 'A', None);
    match result {
        Some(node) => println!("Search ended at: {}", node),
        None => println!("Traversal complete"),
    }
}
```