Contents

Giải Thích Code Sử Dụng Cho Bài Toán Người Nông Dân Qua Sông

Mục Đích

Mã code này giải quyết bài toán về việc di chuyển các thành phần (Nông dân, sói, dê, lúa) qua sông sử dụng thuật toán tìm kiếm theo Chiều Rộng (BFS) và thuật toán tìm kiếm theo Chiều Sâu (DFS).

BFS

Cấu Trúc Code

Lớp State:

Lớp này đại diện cho mỗi trạng thái của bài toán.

Các thuộc tính: **farmer, wolf, goat, rice** biểu thị vị trí của các đối tượng.

Phương thức:

isValid: kiểm tra xem trạng thái có hợp lệ không.

isGoal: kiểm tra xem trạng thái có phải là trạng thái mục tiêu không.

clone: tạo một bản sao của trạng thái.

equals: kiểm tra tính bằng nhau với một đối tượng khác.

hashCode: tạo mã băm cho trạng thái.

toString: cung cấp một chuỗi định dạng.

Lớp BFS:

Bao gồm phương thức main để bắt đầu BFS.

Phương thức bfs:

Thực hiện thuật toán BFS.

Sử dụng một hàng đợi và một danh sách để theo dõi các trạng thái đã duyệt qua.

In ra mỗi trạng thái và dừng lại khi đạt đến trạng thái mục tiêu.

Định Dạng Output

Đối với mỗi bước, code sẽ in ra trạng thái hiện tại. Khi đạt đến trạng thái mục tiêu, xuất thông báo thành công cùng với trạng thái cuối cùng.

Code

```
package Drill.Execise_1;

import java.util.ArrayList;
import java.util.LinkedList;
import java.util.Queue;

class State {
    int farmer;
    int wolf;
    int goat;
    int rice;

    public State(int farmer, int wolf, int goat, int rice) {
        this.farmer = farmer;
        this.wolf = wolf;
        this.goat = goat;
        this.rice = rice;
    }
```

```
    public boolean isValid() {
        // Kiểm tra điều kiện hợp lệ
        if ((wolf == goat && farmer != wolf) || (goat == rice && farmer !=
goat)) {
            return false;
        }
        return true;
    }

    public boolean isGoal() {
        // Kiểm tra điều kiện đạt được mục tiêu
        return farmer == 1 && wolf == 1 && goat == 1 && rice == 1;
    }

    public State clone() {
        return new State(farmer, wolf, goat, rice);
    }

    @Override
    public boolean equals(Object obj) {
        if (this == obj) return true;
        if (obj == null || getClass() != obj.getClass()) return false;
        State state = (State) obj;
        return farmer == state.farmer && wolf == state.wolf && goat ==
state.goat && rice == state.rice;
    }

    @Override
    public int hashCode() {
        return 31 * (31 * (31 * farmer + wolf) + goat) + rice;
    }

    @Override
    public String toString() {
        return String.format("Nông dân: %d, Sói: %d, Dê: %d, Lúa: %d", farmer,
wolf, goat, rice);
    }
}

public class BFS {
    public static void main(String[] args) {
        bfs();
    }

    private static void bfs() {
        Queue<State> queue = new LinkedList<>();
        ArrayList<State> visited = new ArrayList<>();
```

```
        State initialState = new State(0, 0, 0, 0);
        queue.add(initialState);
        visited.add(initialState);

        while (!queue.isEmpty()) {
            State currentState = queue.poll();
            System.out.println(currentState);

            if (currentState.isGoal()) {
                System.out.println("Đã đạt được mục tiêu!");
                break;
            }

            // Thử tất cả các bước có thể từ trạng thái hiện tại
            for (int i = 0; i < 4; i++) {
                State nextState = currentState.clone();
                nextState.farmer = 1 - nextState.farmer;

                switch (i) {
                    case 0:
                        nextState.wolf = 1 - nextState.wolf;
                        break;
                    case 1:
                        nextState.goat = 1 - nextState.goat;
                        break;
                    case 2:
                        nextState.rice = 1 - nextState.rice;
                        break;
                    case 3:
                        nextState.wolf = 1 - nextState.wolf;
                        nextState.goat = 1 - nextState.goat;
                        break;
                }

                if (nextState.isValid() && !visited.contains(nextState)) {
                    queue.add(nextState);
                    visited.add(nextState);
                }
            }
        }
    }

}
```

DFS

Cấu Trúc Code

Lớp State:

Lớp này đại diện cho mỗi trạng thái của bài toán.

Các thuộc tính: **farmer, wolf, goat, rice** biểu thị vị trí của các đối tượng.

Phương thức:

isValid: kiểm tra xem trạng thái có hợp lệ không.

isGoal: kiểm tra xem trạng thái có phải là trạng thái mục tiêu không.

clone: tạo một bản sao của trạng thái.

equals: kiểm tra tính bằng nhau với một đối tượng khác.

hashCode: tạo mã băm cho trạng thái.

toString: cung cấp một chuỗi định dạng.

Lớp DFS:

Bao gồm phương thức main để bắt đầu DFS.

Phương thức dfs:

Thực hiện thuật toán DFS.

Sử dụng một ngăn xếp và một tập hợp để theo dõi các trạng thái đã duyệt qua.

In ra mỗi trạng thái và dừng lại khi đạt đến trạng thái mục tiêu.

Định Dạng Output

Đối với mỗi bước, code sẽ in ra trạng thái hiện tại. Khi đạt đến trạng thái mục tiêu, xuất thông báo thành công cùng với trạng thái cuối cùng.

Code

```
package Drill.Execise_1;

import java.util.ArrayList;
import java.util.LinkedList;
import java.util.Queue;
import java.util.Stack;
import java.util.HashSet;

class State {
    int farmer;
    int wolf;
    int goat;
    int rice;

    public State(int farmer, int wolf, int goat, int rice) {
        this.farmer = farmer;
        this.wolf = wolf;
        this.goat = goat;
        this.rice = rice;
    }

    public boolean isValid() {
        // Kiểm tra điều kiện hợp lệ
        if ((wolf == goat && farmer != wolf) || (goat == rice && farmer !=
goat)) {
            return false;
        }
        return true;
    }

    public boolean isGoal() {
        // Kiểm tra điều kiện đạt được mục tiêu
        return farmer == 1 && wolf == 1 && goat == 1 && rice == 1;
    }

    public State clone() {
        return new State(farmer, wolf, goat, rice);
    }

    @Override
    public boolean equals(Object obj) {
        if (this == obj) return true;
        if (obj == null || getClass() != obj.getClass()) return false;
        State state = (State) obj;
```

```
        return farmer == state.farmer && wolf == state.wolf && goat ==
state.goat && rice == state.rice;
    }

    @Override
    public int hashCode() {
        return 31 * (31 * (31 * farmer + wolf) + goat) + rice;
    }

    @Override
    public String toString() {
        return String.format("Nông dân: %d, Sói: %d, Dê: %d, Lúa: %d", farmer,
wolf, goat, rice);
    }
}
public class DFS {
    public static void main(String[] args) {
        dfs();
    }

    private static void dfs() {
        Stack<State> stack = new Stack<>();
        HashSet<State> visited = new HashSet<>();

        State initialState = new State(0, 0, 0, 0);
        stack.push(initialState);
        visited.add(initialState);

        while (!stack.isEmpty()) {
            State currentState = stack.pop();
            System.out.println(currentState);

            if (currentState.isGoal()) {
                System.out.println("Đã đạt được mục tiêu!");
                break;
            }

            // Thử tất cả các bước có thể từ trạng thái hiện tại
            for (int i = 0; i < 4; i++) {
                State nextState = currentState.clone();
                nextState.farmer = 1 - nextState.farmer;

                switch (i) {
                    case 0:
                        nextState.wolf = 1 - nextState.wolf;
                        break;
                    case 1:
                        nextState.goat = 1 - nextState.goat;
```

```
                        break;
                    case 2:
                        nextState.rice = 1 - nextState.rice;
                        break;
                    case 3:
                        nextState.wolf = 1 - nextState.wolf;
                        nextState.goat = 1 - nextState.goat;
                        break;
                }

                if (nextState.isValid() && !visited.contains(nextState)) {
                    stack.push(nextState);
                    visited.add(nextState);
                }
            }
        }
    }
}
```