Contents

Giải Bài toán Bình Nước bằng Thuật toán A*

Mục Đích

Mã code này giải quyết bài toán đong bình nước bằng thuật toán A*

Cấu Trúc Code

1. **Class TrangThai**:
    1. Mô tả từng trạng thái của bình nước.
    2. Thuộc tính: binhA, binhB, binhC biểu diễn dung tích của từng bình.
    3. Phương thức:
        1. equals: So sánh xem hai trạng thái có bằng nhau không.
        2. hashCode: Tạo mã băm cho trạng thái.
2. **Class Node implements Comparable**:
    1. Biểu diễn một nút trong đồ thị tìm kiếm A*.
    2. Thuộc tính: trangThai, cha, chiPhi (g(n)), heuristic (h(n)).
    3. Phương thức:
        1. compareTo: So sánh hai nút dựa trên chi phí tổng (g(n) + h(n)).
3. **Class AStar**:

    1. Chứa phương thức main và triển khai thuật toán A*.
    2. Các hằng số DUNG_TICH_A, DUNG_TICH_B, DUNG_TICH_C là dung tích của các bình nước.
4. **Phương thức main**:
    1. Khởi tạo trạng thái ban đầu và mục tiêu.
    2. Gọi phương thức thucHienThuatToanAStar để thực hiện thuật toán.
5. **Phương thức thucHienThuatToanAStar**:
    1. Sử dụng PriorityQueue để quản lý tập hợp các nút theo thứ tự ưu tiên.
    2. Sử dụng HashSet để lưu trạng thái đã xét.
    3. Hiển thị chi tiết từng bước thực hiện thuật toán, bao gồm trạng thái đang xét, chi phí, và heuristic.
    4. Khi tìm ra giải pháp, hiển thị đường đi giải pháp.
6. **Phương thức sinhTrangThaiLienKe**:
    1. Sinh ra các trạng thái liên kết từ một trạng thái cho trước.
7. **Phương thức themTrangThaiNeuChuaCo**:
    1. Thêm một trạng thái vào danh sách nếu nó chưa tồn tại.
8. **Phương thức heuristic**:
    1. Sử dụng khoảng cách Manhattan để tính heuristic giữa hai trạng thái.
9. **Phương thức inLoiGiai**:
    1. In ra đường đi giải pháp từ trạng thái ban đầu đến trạng thái mục tiêu.

Định Dạng Kết Quả Output

- Mỗi bước di chuyển sẽ hiển thị trạng thái đang xét, chi phí, và heuristic.
- Khi tìm ra giải pháp, hiển thị đường đi giải pháp từ trạng thái ban đầu đến trạng thái mục tiêu.

Code

```
package Drill.Execise_2;
import java.util.*;

class TrangThai {
    int binhA, binhB, binhC;

    public TrangThai(int a, int b, int c) {
        this.binhA = a;
        this.binhB = b;
        this.binhC = c;
    }

    @Override
    public boolean equals(Object obj) {
        if (obj == this)
            return true;
        if (!(obj instanceof TrangThai))
            return false;
        TrangThai trangThai = (TrangThai) obj;
        return this.binhA == trangThai.binhA && this.binhB == trangThai.binhB
&& this.binhC == trangThai.binhC;
    }

    @Override
    public int hashCode() {
        return Objects.hash(binhA, binhB, binhC);
    }
}

class Node implements Comparable<Node> {
    TrangThai trangThai;
    Node cha;
    int chiPhi; // g(n)
    int heuristic; // h(n)

    public Node(TrangThai trangThai, Node cha, int chiPhi, int heuristic) {
```

```
        this.trangThai = trangThai;
        this.cha = cha;
        this.chiPhi = chiPhi;
        this.heuristic = heuristic;
    }

    @Override
    public int compareTo(Node other) {
        return Integer.compare(this.chiPhi + this.heuristic, other.chiPhi +
other.heuristic);
    }
}

public class AStar {
    private static final int DUNG_TICH_A = 3;
    private static final int DUNG_TICH_B = 5;
    private static final int DUNG_TICH_C = 8;

    public static void main(String[] args) {
        TrangThai trangThaiBanDau = new TrangThai(0, 0, 8); // Trạng thái ban
đầu: A = 0, B = 0, C = 8
        TrangThai trangThaiDich = new TrangThai(0, 4, 4); // Trạng thái đích:
A = 0, B = 4, C = 4

        thucHienThuatToanAStar(trangThaiBanDau, trangThaiDich);
    }

    private static void thucHienThuatToanAStar(TrangThai trangThaiBanDau,
TrangThai trangThaiDich) {
        PriorityQueue<Node> openSet = new PriorityQueue<>();
        Set<TrangThai> closedSet = new HashSet<>();

        Node nodeBanDau = new Node(trangThaiBanDau, null, 0,
heuristic(trangThaiBanDau, trangThaiDich));
        openSet.add(nodeBanDau);

        while (!openSet.isEmpty()) {
            Node nodeHienTai = openSet.poll();
            TrangThai trangThaiHienTai = nodeHienTai.trangThai;

            System.out.println("\nTrạng thái: " + trangThaiHienTai.binhA + " "
+ trangThaiHienTai.binhB + " " + trangThaiHienTai.binhC);
            System.out.println("Chi phí: " + nodeHienTai.chiPhi);
            System.out.println("Heuristic: " + nodeHienTai.heuristic);

            if (trangThaiHienTai.equals(trangThaiDich)) {
                System.out.println("Đã đạt được trạng thái đích!");
                inLoiGiai(nodeHienTai);
```

```
                return;
            }

            closedSet.add(trangThaiHienTai);

            List<TrangThai> trangThaiLienKe =
sinhTrangThaiLienKe(trangThaiHienTai);
            for (TrangThai lienKe : trangThaiLienKe) {
                if (closedSet.contains(lienKe)) {
                    continue;
                }

                int chiPhiMoi = nodeHienTai.chiPhi + 1; // Giả sử chi phí =
                int heuristic = heuristic(lienKe, trangThaiDich);
                Node nodeLienKe = new Node(lienKe, nodeHienTai, chiPhiMoi,
heuristic);

                if (!openSet.contains(nodeLienKe)) {
                    openSet.add(nodeLienKe);
                }
            }
        }

        System.out.println("Không tìm thấy giải pháp.");
    }

    private static List<TrangThai> sinhTrangThaiLienKe(TrangThai
trangThaiHienTai) {
        List<TrangThai> trangThaiLienKe = new ArrayList<>();

        // Đổ đầy A
        themTrangThaiNeuChuaCo(trangThaiLienKe, new TrangThai(DUNG_TICH_A,
trangThaiHienTai.binhB, trangThaiHienTai.binhC), "Đổ đầy A");
        // Đổ đầy B
        themTrangThaiNeuChuaCo(trangThaiLienKe, new
TrangThai(trangThaiHienTai.binhA, DUNG_TICH_B, trangThaiHienTai.binhC), "Đổ
đầy B");
        // Đổ đầy C
        themTrangThaiNeuChuaCo(trangThaiLienKe, new
TrangThai(trangThaiHienTai.binhA, trangThaiHienTai.binhB, DUNG_TICH_C), "Đổ
đầy C");

        // Rót từ A sang B
        int rotAB = Math.min(trangThaiHienTai.binhA, DUNG_TICH_B -
trangThaiHienTai.binhB);
        themTrangThaiNeuChuaCo(trangThaiLienKe, new
TrangThai(trangThaiHienTai.binhA - rotAB, trangThaiHienTai.binhB + rotAB,
trangThaiHienTai.binhC), "Rót từ A sang B");
```

```
        // Rót từ A sang C
        int rotAC = Math.min(trangThaiHienTai.binhA, DUNG_TICH_C -
trangThaiHienTai.binhC);
        themTrangThaiNeuChuaCo(trangThaiLienKe, new
TrangThai(trangThaiHienTai.binhA - rotAC, trangThaiHienTai.binhB,
trangThaiHienTai.binhC + rotAC), "Rót từ A sang C");

        // Rót từ B sang A
        int rotBA = Math.min(trangThaiHienTai.binhB, DUNG_TICH_A -
trangThaiHienTai.binhA);
        themTrangThaiNeuChuaCo(trangThaiLienKe, new
TrangThai(trangThaiHienTai.binhA + rotBA, trangThaiHienTai.binhB - rotBA,
trangThaiHienTai.binhC), "Rót từ B sang A");

        // Rót từ B sang C
        int rotBC = Math.min(trangThaiHienTai.binhB, DUNG_TICH_C -
trangThaiHienTai.binhC);
        themTrangThaiNeuChuaCo(trangThaiLienKe, new
TrangThai(trangThaiHienTai.binhA, trangThaiHienTai.binhB - rotBC,
trangThaiHienTai.binhC + rotBC), "Rót từ B sang C");

        // Rót từ C sang A
        int rotCA = Math.min(trangThaiHienTai.binhC, DUNG_TICH_A -
trangThaiHienTai.binhA);
        themTrangThaiNeuChuaCo(trangThaiLienKe, new
TrangThai(trangThaiHienTai.binhA + rotCA, trangThaiHienTai.binhB,
trangThaiHienTai.binhC - rotCA), "Rót từ C sang A");

        // Rót từ C sang B
        int rotCB = Math.min(trangThaiHienTai.binhC, DUNG_TICH_B -
trangThaiHienTai.binhB);
        themTrangThaiNeuChuaCo(trangThaiLienKe, new
TrangThai(trangThaiHienTai.binhA, trangThaiHienTai.binhB + rotCB,
trangThaiHienTai.binhC - rotCB), "Rót từ C sang B");

        return trangThaiLienKe;
    }

    private static void themTrangThaiNeuChuaCo(List<TrangThai>
trangThaiLienKe, TrangThai trangThaiMoi, String hanhDong) {
        if (!trangThaiLienKe.contains(trangThaiMoi)) {
            trangThaiLienKe.add(trangThaiMoi);
            System.out.println("Thực hiện: " + hanhDong + " --> Trạng thái
mới: " + trangThaiMoi.binhA + " " + trangThaiMoi.binhB + " " +
trangThaiMoi.binhC);
        }
    }
```

```
    private static int heuristic(TrangThai hienTai, TrangThai mucTieu) {
         // Định nghĩa hàm heuristic theo khoảng cách Manhattan
        return Math.abs(hienTai.binhA - mucTieu.binhA) +
Math.abs(hienTai.binhB - mucTieu.binhB) + Math.abs(hienTai.binhC -
mucTieu.binhC);
    }

    private static void inLoiGiai(Node nodeMucTieu) {
        List<TrangThai> duongDi = new ArrayList<>();
        Node nodeHienTai = nodeMucTieu;

        while (nodeHienTai != null) {
            duongDi.add(nodeHienTai.trangThai);
            nodeHienTai = nodeHienTai.cha;
        }

        Collections.reverse(duongDi);

        System.out.println("Đường đi giải pháp:");
        for (TrangThai trangThai : duongDi) {
            System.out.println(trangThai.binhA + " " + trangThai.binhB + " " +
trangThai.binhC);
        }
    }
}
```