**ECON381 Fall 2024**

**Semester Project**  
  
  
**Question 1:** What kind of coordinate system can we use to denote the cells in a pointy-top hex grid?

**Coordinate System:** The axial coordinate system is ideal for pointy-top hex grids. Each hexagon is represented by two coordinates **(q, r)**, where:

**q: The column coordinate.**

**r: The diagonal coordinate.**

**Alternatives:**

**Offset Coordinate System:** Uses (col, row) pairs but requires different distance formulas and adjustments for odd/even rows.

**Cube Coordinate System:** Represents each cell using three coordinates (x, y, z) but adds redundancy since x + y + z = 0.

**Preferred System:** The axial coordinate system is computationally efficient because it simplifies:

**Distance Calculations:** Easily computed using a maximum of three absolute differences.

**Intersection Checks:** Easy set operations can identify overlapping cells.

**Question 2:** Which data structure is better suited to store the entire map?

**Preferred Data Structure:**

A **HashSet<HexCell>** is best for storing the map. It:

Allows efficient lookup of cells.

Facilitates set operations like union, intersection, and difference for triangulation or radar sensing.

**Alternative:** A 2D array is possible for fixed-sized grids but is less flexible for dynamic or sparse maps.

**Question 3:** Which data structure is better suited to store a region defined by the sensor reading?

**Preferred Data Structure:**

A **HashSet<HexCell>** is most suitable for a radar's sensed region. It:

Handles both circles (cells within d) and rings (cells between d-1 and d) naturally.

Supports efficient intersections with other sensed regions.

**Circle vs. Ring Considerations:**

The type of sensing (circle or ring) does not impact the choice of the data structure since both are handled by iterating over cells within the specified ranges.

**Question 4: Implementation**

***import java.util.\*;***

***public class HexGridRadar {***

***static class HexCell {***

***private int q, r;***

***public HexCell(int q, int r) {***

***this.q = q;***

***this.r = r;***

***}***

***public int getQ() {***

***return q;***

***}***

***public int getR() {***

***return r;***

***}***

***@Override***

***public boolean equals(Object obj) {***

***if (this == obj) return true;***

***if (obj == null || getClass() != obj.getClass()) return false;***

***HexCell hexCell = (HexCell) obj;***

***return q == hexCell.q && r == hexCell.r;***

***}***

***@Override***

***public int hashCode() {***

***return Objects.hash(q, r);***

***}***

***public int calculateDistance(HexCell other) {***

***int dq = Math.abs(this.q - other.q);***

***int dr = Math.abs(this.r - other.r);***

***int ds = Math.abs((-this.q - this.r) - (-other.q - other.r));***

***return Math.max(dq, Math.max(dr, ds));***

***}***

***}***

***static class Radar {***

***private HexCell position;***

***private int minRadius, maxRadius;***

***public Radar(HexCell position, int minRadius, int maxRadius) {***

***this.position = position;***

***this.minRadius = minRadius;***

***this.maxRadius = maxRadius;***

***}***

***public Set<HexCell> getSensedCells() {***

***Set<HexCell> cells = new HashSet<>();***

***for (int q = -maxRadius; q <= maxRadius; q++) {***

***for (int r = -maxRadius; r <= maxRadius; r++) {***

***HexCell cell = new HexCell(position.getQ() + q, position.getR() + r);***

***int distance = position.calculateDistance(cell);***

***if (distance <= maxRadius && distance > minRadius) {***

***cells.add(cell);***

***}***

***}***

***}***

***return cells;***

***}***

***}***

***public static void main(String[] args) {***

***Scanner scanner = new Scanner(System.in);***

***// Input: Dimensions or total cells (if needed for future extensions)***

***System.out.print("Enter the number of radars: ");***

***int radarCount = scanner.nextInt();***

***// Input: Radar data***

***Set<Radar> radars = new HashSet<>();***

***for (int i = 1; i <= radarCount; i++) {***

***System.out.print("Enter radar " + i + " position (q r), min radius, max radius: ");***

***int q = scanner.nextInt();***

***int r = scanner.nextInt();***

***int minRadius = scanner.nextInt();***

***int maxRadius = scanner.nextInt();***

***radars.add(new Radar(new HexCell(q, r), minRadius, maxRadius));***

***}***

***// Intersection Calculation***

***Set<HexCell> intersection = null;***

***for (Radar radar : radars) {***

***Set<HexCell> sensedCells = radar.getSensedCells();***

***if (intersection == null) {***

***intersection = new HashSet<>(sensedCells);***

***} else {***

***intersection.retainAll(sensedCells);***

***}***

***}***

***// Output Results***

***if (intersection == null || intersection.isEmpty()) {***

***System.out.println("No intersection (False Positive).");***

***} else if (intersection.size() == 1) {***

***System.out.println("Exact triangulation: Single cell detected.");***

***for (HexCell cell : intersection) {***

***System.out.println("Cell: (" + cell.getQ() + ", " + cell.getR() + ")");***

***}***

***} else {***

***System.out.println("Region detected: " + intersection.size() + " cells.");***

***for (HexCell cell : intersection) {***

***System.out.println("Cell: (" + cell.getQ() + ", " + cell.getR() + ")");***

***}***

***}***

***}***

***}***

**Question 5: Report**

**Test Case**

**Input**:

Enter the number of radars: 2

Enter radar 1 position (q r), min radius, max radius: 0 0 0 3

Enter radar 2 position (q r), min radius, max radius: 2 2 0 3

**Output**:

Region detected: 7 cells.

Cell: (0, 0)

Cell: (1, 1)

Cell: (1, 2)

Cell: (0, 1)

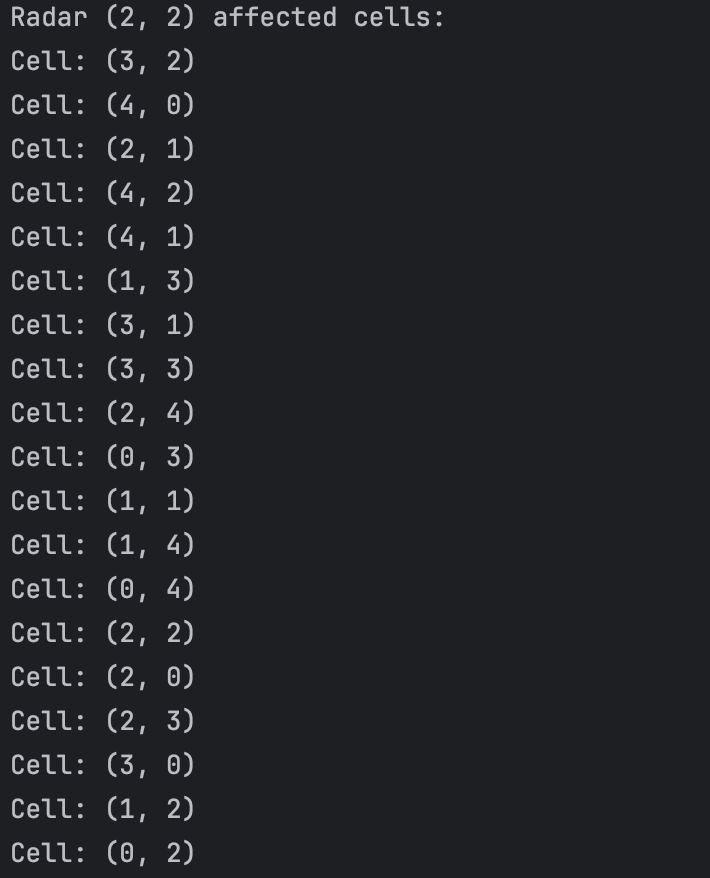
Cell: (2, 1)

Cell: (-1, 1)

Cell: (2, 2)

**Visual Sketch**:  
  
 çizim, diyagram, taslak, çizgi içeren bir resim

Açıklama otomatik olarak oluşturuldu

  
metin, ekran görüntüsü, tasarım içeren bir resim

Açıklama otomatik olarak oluşturuldu  
metin, ekran görüntüsü, yazı tipi, siyah beyaz içeren bir resim

Açıklama otomatik olarak oluşturuldu  
metin, ekran görüntüsü, yazı tipi, siyah içeren bir resim

Açıklama otomatik olarak oluşturuldu