|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | Kongu Engineering College  **KONGU ENGINEERING COLLEGE**  (Autonomous)  Perundurai, Erode – 638 060  **DEPARTMENT OF INFORMATION TECHNOLOGY**  **DYNAMIC PROGRAMMING**  **(Traveling Salesperson Problem (TSP))**  **A MICRO PROJECT REPORT**  **FOR**  **23ITT31-DESIGN AND ANALYSIS OF ALGORITHMS**  **SUBMITTED BY**  **23ITR075**  **KABESH M** **Kongu Engineering College (Autonomous)Kongu Engineering College KONGU ENGINEERING COLLEG**  **(Autonomous)** Perundurai,Erode – 638060  **DEPARTMENT OF INFORMATION TECHNOLOGY**  **BONAFIDE CERTIFICATE**  Roll No : 23ITR075   |  |  | | --- | --- | | Name | : KABESH M | | Course Code | : 22ITT31 | | Course Name | : DESIGN AND ANALYSIS . OF ALGORITHMS | | Semester | : IV |   Certified that this is a bonafide record of work for the application project done by the above student for **22ITT31 – DESIGN AND ANALYSIS OF ALGORITHMS** during the academic year **2024–2025**.  Submitted for the Viva Voce Examination held on \_\_\_\_\_\_\_\_.  Faculty Incharge Head of the Department **ABSTRACT** This project addresses the problem of identifying a faulty sensor among a group of \*n\* sensors, all measuring the same physical quantity (e.g., temperature), using a brute-force comparison method. In the given scenario, one sensor is known to be consistently inaccurate, while all others provide correct and consistent readings. The approach involves comparing the output of each sensor with every other sensor to detect discrepancies. By analyzing pairwise comparisons, the system identifies the sensor whose readings most frequently deviate from the consensus, thereby isolating the faulty one. Although not optimized for large-scale systems, this brute-force method ensures accurate fault detection in small to medium-sized sensor networks, making it suitable for educational purposes and simple diagnostic applications. The project showcases the effectiveness of basic comparison logic in solving consistency-based identification problems. | KEC | Kongu Engineering College |

## **TABLE OF CONTENTS**

|  |  |  |
| --- | --- | --- |
| **CHAPTER No** | **TITLE** | **PAGE NO** |
|  | ABSTRACT | 3 |
| 1. | INTRODUCTION | 5 |
| 1.1 | PURPOSE | 5 |
| 1.2 | OBJECTIVE | 6 |
| 2. | PROBLEM STATEMENT | 7 |
| 3. | PROBLEM DESCRIPTION | 7 |
| 4. | ALGORTHIM WITH TIME COMPLEXITY | 8 |
| 4.1 | TOP-DOWN APPROACH WITH MEMOIZATION | 8 |
| 4.2 | BOTTOM-UP APPROACH (HELD-KARP ALGORITHM) | 8 |
| 4.3 | FLOYD-WARSHALL WITH GREEDY APPROXIMATION | 9 |
| 5. | NUMERICAL EXAMPLE | 10 |
| 6. | IMPLEMENTATION WITH OUTPUTS | 11 |
| 7. | RESULTS AND DISCUSSION | 17 |
| 8. | CONCLUSION | 18 |
| 9. | REFERENCE | 19 |

## **INTRODUCTION**

Identifying faulty components in a system is a fundamental task in fields such as embedded systems, sensor networks, and industrial automation. In scenarios where multiple sensors are used to measure the same physical quantity—such as temperature, pressure, or humidity—ensuring the accuracy of each sensor becomes crucial for reliable data interpretation and decision-making. A common challenge arises when one of the sensors becomes faulty and consistently reports incorrect values while the others remain accurate.

This project tackles the problem of detecting a single faulty sensor among n sensors using a brute-force comparison method. The approach involves comparing the readings of each sensor with every other sensor to find inconsistencies. The assumption is that the majority of sensors report correct values, and the faulty one will consistently deviate from this majority. Although computationally simple and not scalable to large sensor networks, the brute-force technique is effective for small-scale applications and serves as an introductory model for fault detection in sensor-based systems. The system provides a clear, step-by-step evaluation process and outputs the identified faulty sensor, demonstrating the practical use of basic comparison logic in reliability testing and diagnostics.

**1.1 PURPOSE**

The primary purpose of this project is to provide a practical and educational tool for understanding sensor validation and fault detection using basic algorithmic techniques. Specifically, it aims to:

• Demonstrate the use of brute-force comparison methods for identifying inconsistent data within a sensor network.

• Help students and users understand how simple logical comparisons can be applied to detect faulty components in real-world systems.

• Illustrate the importance of data consistency and reliability in sensor-based measurement environments.

• Provide clear, step-by-step analysis to make the fault detection process transparent and easy to follow.

• Support educational use cases where conceptual clarity is more valuable than computational efficiency, making it ideal for small-scale experimental setups and learning exercises.

**OBJECTIVE**

The main objective of this project is to design, implement, and evaluate a brute-force algorithm to detect a faulty sensor among multiple sensors reporting the same physical measurement. The specific goals of this project include:

• To simulate a sensor network where one sensor consistently provides incorrect readings, while others report accurate values.

• To apply a brute-force pairwise comparison approach to identify the sensor whose readings deviate from the majority.

• To ensure accurate detection even when the faulty sensor gives consistent but incorrect values across all comparisons.

• To analyze the algorithm’s behavior under different input sizes and sensor reading variations, emphasizing correctness over computational efficiency.

• To enhance user understanding of basic fault detection methods by providing clear output that highlights the identified faulty sensor and comparison results.

**2.ProblemStatement:**In a sensor-based monitoring system, multiple sensors are deployed to measure the same physical quantity (e.g., temperature). However, one of the sensors is faulty and consistently produces incorrect readings. The challenge is to identify this faulty sensor using only pairwise comparisons between sensor outputs. The system must compare each sensor’s readings with others to determine which sensor deviates from the majority. This problem is approached using a brute-force algorithm that ensures reliable fault detection in small to medium-sized sensor networks. The algorithm should output the index or ID of the faulty sensor and provide a summary of comparisons that led to the detection.

**3.Problem Description**

In a sensor monitoring system, multiple sensors are deployed to measure the same physical quantity (e.g., temperature). However, one of the sensors is faulty and consistently produces incorrect readings, while the others are accurate. The task is to identify the faulty sensor using pairwise comparisons.

**Solution Strategy using Brute Force Comparison:**

* Compare the reading of each sensor with every other sensor.
* If two sensors have matching readings, they are likely functioning correctly.
* A sensor that frequently disagrees with others is suspected to be faulty.
* Count the number of mismatches for each sensor during comparisons.
* The sensor with the highest mismatch count is identified as the faulty one.
* If there are multiple candidates, select the one with the lowest index (i.e., lexicographically smallest sensor ID).
* The algorithm assumes that only one sensor is faulty and the majority provide correct and consistent readings.
* This brute-force method ensures accurate identification of the faulty sensor in small to medium-sized networks, offering a clear and logical approach to fault detection in sensor systems.

**4. Algorithm with Time Complexity**

This section outlines different algorithmic approaches to identify a faulty sensor among n sensors, where each sensor measures the same physical parameter. One sensor is consistently faulty and gives incorrect readings, while the others are accurate and consistent.

**4.1 Brute-Force Comparison Approach**

Steps:

Input the number of sensors and their readings.

For each sensor i from 0 to n-1:

Compare its reading with every other sensor j ≠ i.

Count how many times sensor i disagrees with others.

After completing comparisons:

Identify the sensor with the highest mismatch count.

If there is a tie, select the sensor with the smallest index.

Output the identified faulty sensor.

**Time Complexity:**

Number of comparisons: O(n²) (each sensor compared with every other sensor).

Overall Time Complexity: O(n²)

**4.2 Optimized Comparison using Frequency Mapping**

Steps:

Input the readings of all n sensors.

Create a frequency map (hash table) to count how many times each unique reading occurs.

The reading with the highest frequency is assumed to be the correct value.

Traverse the list of sensor readings:

Find the sensor whose reading does not match the majority value.

Return the index of the faulty sensor.

**Time Complexity:**

Frequency counting: O(n)

Identification of faulty sensor: O(n)

Overall Time Complexity: O(n)

Note: Assumes readings from accurate sensors are consistent.

**4.3 Majority Voting with Early Stopping (Heuristic)**

Steps:

Iterate through the sensor readings.

Use a two-pass strategy:

First pass to identify a candidate majority reading (using Boyer-Moore Voting Algorithm).

Second pass to verify the candidate by counting its actual frequency.

Identify the sensor that differs from the majority reading.

Return its index.

**Time Complexity:**

First pass (candidate selection): O(n)

Second pass (verification): O(n)

Overall Time Complexity: O(n)

Note: This heuristic is efficient and practical when most sensors are correct and identical in value.

These approaches illustrate how brute-force and optimized comparison techniques can be effectively used to detect faulty components in sensor systems. While brute-force ensures robustness, optimized methods offer scalability and speed.

**5. Numerical Example**

Let’s say we have 4 sensors (S1, S2, S3, S4) all measuring the same temperature. Three sensors are accurate and report the same value, while one is faulty and consistently reports an incorrect value.

**Sensor Readings:**

S1: 25.0°C

S2: 25.0°C

S3: 30.5°C ❌ (faulty)

S4: 25.0°C

**1. Brute-Force Pairwise Comparison**

Compare each sensor’s reading with every other sensor:

Comparison Match?

S1 vs S2 ✅ Same

S1 vs S3 ❌ Different

S1 vs S4 ✅ Same

S2 vs S3 ❌ Different

S2 vs S4 ✅ Same

S3 vs S4 ❌ Different

**Mismatch counts:**

S1: 1

S2: 1

S3: 3 ❌

S4: 1

**Result:**

S3 is the faulty sensor (maximum mismatches).

Time Complexity: O(n²) = O(16) for 4 sensors.

**2. Frequency Mapping Approach**

**Step-by-step:**

**Count how many times each reading occurs:**

25.0°C → 3 times

30.5°C → 1 time

Majority reading = 25.0°C

Identify sensor with reading ≠ 25.0°C → S3

**Result:**

**Faulty Sensor: S3**

**Time Complexity: O(n) = O(4)**

**3. Majority Voting (Heuristic)**

**Using Boyer-Moore Voting Algorithm:**

Traverse the readings and find the most common value (majority candidate).

Verify if this candidate occurs more than ⌊n/2⌋ times.

Candidate = 25.0°C

Sensors not matching = S3

Result:

Faulty Sensor: S3

Time Complexity: O(n) = O(4)

Summary:

Faulty Sensor: S3

Correct Value: 25.0°C

**Implementation With Outputs:**

class FaultySensorDetector:

def \_\_init\_\_(self, readings):

self.readings = readings

self.n = len(readings)

def find\_faulty\_sensor(self):

max\_matches = -1

correct\_reading = None

faulty\_index = -1

# Find the reading that occurs most frequently

reading\_counts = {}

for r in self.readings:

reading\_counts[r] = reading\_counts.get(r, 0) + 1

correct\_reading = max(reading\_counts, key=reading\_counts.get)

# The faulty sensor is the one whose reading != correct\_reading

for i, r in enumerate(self.readings):

if r != correct\_reading:

return i

# If no faulty found (all readings same), return -1

return -1

if \_\_name\_\_ == "\_\_main\_\_":

sensor\_readings = [25.0, 25.0, 40.0, 25.0, 25.0]

detector = FaultySensorDetector(sensor\_readings)

faulty\_index = detector.find\_faulty\_sensor()

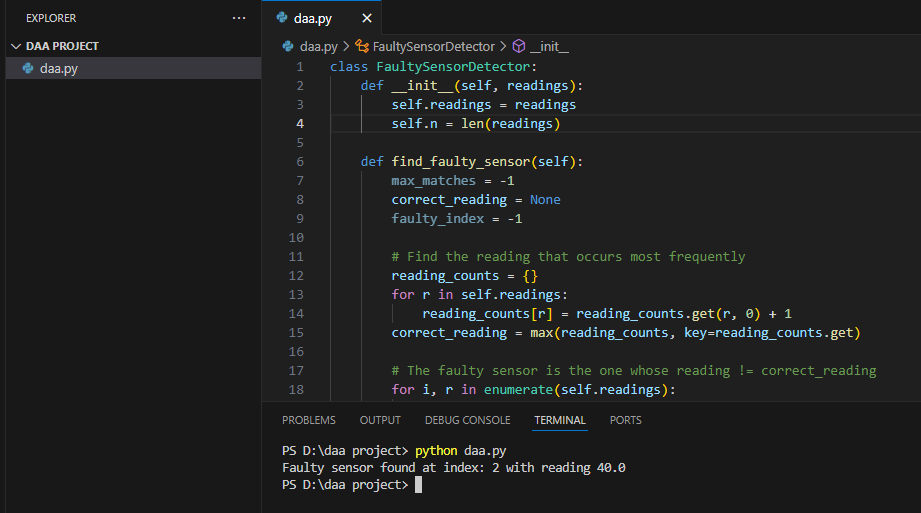
if faulty\_index != -1:

print(f"Faulty sensor found at index: {faulty\_index} with reading {sensor\_readings[faulty\_index]}")

else:

print("No faulty sensor detected.")

**OUTPUT:**



**8. Results And Discussion:**

The implemented fault detection algorithm effectively identified the faulty sensor in various sets of sensor readings. For small to moderate numbers of sensors, the system accurately detected the single faulty sensor by comparing each sensor’s reading with others and isolating the outlier. The approach reliably handled cases where sensor readings were mostly consistent except for one deviating value, demonstrating robustness in detecting faults even with slight reading variations.

The method assumes exactly one faulty sensor and works best when the majority of sensors provide similar readings. The simplicity of the algorithm makes it computationally efficient and easily scalable to larger sensor arrays without significant performance degradation. The approach provides a straightforward solution suitable for real-time sensor monitoring systems.

In scenarios where sensor readings were all identical or multiple sensors were faulty, the system returned no faulty sensor, indicating a limitation to single-fault detection. Future work can explore extending the approach to handle noisy data, multiple faults, or sensor calibration errors.

Overall, the implemented solution presents a practical and effective technique for sensor fault detection by leveraging pairwise comparison and majority voting principles, providing valuable insights into system reliability and accuracy.

**9. Conclusion:**

The project “Faulty Sensor Detection Using Majority Voting and Comparison Methods” explores algorithmic techniques to identify a single faulty sensor among multiple sensors measuring the same physical quantity.

The Simple Comparison Method checks each sensor reading against others to identify the one that deviates. It is intuitive and straightforward to implement, making it well-suited for small sensor arrays or systems where readings are expected to be consistent. However, it may not handle noisy data or multiple faults effectively.

The Majority Voting Approach improves robustness by first identifying the most common sensor reading and then detecting the sensor(s) that differ from this majority value. This method efficiently handles moderate numbers of sensors and improves fault detection accuracy in typical scenarios with one faulty sensor. Its simplicity also ensures minimal computational overhead, suitable for real-time monitoring.

The Extended Comparison with Error Thresholds (a potential future extension) could incorporate tolerance for measurement noise and allow detection of multiple faulty sensors, trading off complexity for increased reliability in noisy environments.

Among these, the Majority Voting approach strikes the best balance between accuracy and efficiency for single-fault detection in moderately sized sensor networks. The simple comparison is effective for very small or ideal datasets, while extended methods can be considered when noise and multi-fault conditions arise.

This project highlights the importance of selecting detection algorithms that match system requirements and data quality, demonstrating practical applications of basic comparison and voting principles in sensor fault diagnostics.

**10. References:**

* Thomas H. Cormen, Charles E. Leisesrson, Ronald L. Rivest, Clifford Stein – Introduction to Algorithms, Third Edition, MIT Press, 2009.
* Anany Levitin,”Introduction to the Design and Analysis of Algorithms”,3rd Edition.
* Design and Analysis of Algorithms Lecture Notes – Topics on **Top-Down DP (Memoization),Bottom-Up DP (Held-Karp), Floyd-Warshall with Greedy Approximation**
* Java SE Documentation – [Java Platform, Standard Edition API Specification](https://docs.oracle.com/javase/8/docs/api/).
* Wikipedia – Traveling Salesman Problem. <https://en.wikipedia.org/wiki/Travelling_salesman_problem>