K. J. Somaiya College of Engineering, Mumbai-77
(A Constituent College of Somaiya Vidyavihar University)

Roll No.: 16010123038 16010123037
Name of the student: Amrit Nigam Ambuj Rai
Div: A2 batch
Branch: Computer Engineering
Group - G19
IA No: IA2
Date:14-10-24
Subject: Discrete Mathematics

TITLE: Set Theory
AIM: To implement To check refelxsive , symmetric , transitive set

Literature survey/Theory:

Introduction: In set theory and discrete mathematics, a **relation** is a set of ordered pairs. Relations can have various properties that define their behavior, such as being reflexive, symmetric, or transitive. Understanding these properties is crucial in fields such as database theory, graph theory, logic, and automata theory. These properties also play an important role in mathematical proofs, algorithms, and decision-making systems.

Reflexive, Symmetric, and Transitive Properties:

1. **Reflexive Relation:** A relation RRR on a set AAA is said to be **reflexive** if every element of the set is related to itself. Mathematically, a relation RRR is reflexive if for all a∈Aa \in Aa∈A, the pair (a,a)∈R(a, a) \in R(a,a)∈R.
   - **Example:** Consider a set A={1,2,3}A = \{1, 2, 3\}A={1,2,3}. A relation RRR on AAA is reflexive if it contains pairs (1,1),(2,2),(3,3)(1, 1), (2, 2), (3, 3)(1,1),(2,2),(3,3), i.e., each element is related to itself.
   - **Application:** Reflexivity is important in areas like logic and theoretical computer science. For example, in database systems, a reflexive relation can represent hierarchical structures where every node is accessible to itself.
2. **Symmetric Relation:** A relation RRR is **symmetric** if for all elements a,b∈Aa, b \in Aa,b∈A, whenever (a,b)∈R(a, b) \in R(a,b)∈R, it follows that (b,a)∈R(b, a) \in R(b,a)∈R. In other words, if one element is related to another, then the reverse must also hold true.

- **Example:** For the same set A={1,2,3}A = \{1, 2, 3\}A={1,2,3}, a relation RRR is symmetric if, whenever (1,2)∈R(1, 2) \in R(1,2)∈R, then (2,1)∈R(2, 1) \in R(2,1)∈R.
- **Application:** Symmetric relations are essential in fields like social network analysis, where mutual relationships (e.g., friendships) are modeled using symmetric relations. It also applies in communication networks where two-way communication is possible.

3. **Transitive Relation:** A relation RRR is **transitive** if for all elements a,b,c∈Aa, b, c \in Aa,b,c∈A, whenever (a,b)∈R(a, b) \in R(a,b)∈R and (b,c)∈R(b, c) \in R(b,c)∈R, it implies that (a,c)∈R(a, c) \in R(a,c)∈R. Transitivity reflects the idea of chaining relationships together.
    - **Example:** In a set A={1,2,3}A = \{1, 2, 3\}A={1,2,3}, if (1,2)∈R(1, 2) \in R(1,2)∈R and (2,3)∈R(2, 3) \in R(2,3)∈R, then (1,3)(1, 3)(1,3) must also be in RRR for the relation to be transitive.
    - **Application:** Transitivity is widely used in graph theory (where transitive closure helps find paths between nodes) and in reasoning systems (where if AAA implies BBB, and BBB implies CCC, then AAA implies CCC).

Mathematical Concept/Algorithms:

1. Reflexive Relation: A relation RRR on set AAA is reflexive if every element is related to itself, i.e., (a,a)∈R(a, a) \in R(a,a)∈R for all a∈Aa \in Aa∈A.
    - Adjacency Matrix: Reflexive relations have 1s on the diagonal.
2. Symmetric Relation: A relation is symmetric if (a,b)∈R(a, b) \in R(a,b)∈R implies (b,a)∈R(b, a) \in R(b,a)∈R for all a,b∈Aa, b \in Aa,b∈A.
    - Adjacency Matrix: The matrix is symmetric, meaning M[i][j]=M[j][i]M[i][j] = M[j][i]M[i][j]=M[j][i].
3. Transitive Relation: A relation is transitive if (a,b)∈R(a, b) \in R(a,b)∈R and (b,c)∈R(b, c) \in R(b,c)∈R imply (a,c)∈R(a, c) \in R(a,c)∈R.
    - Warshall's Algorithm: Computes the transitive closure of a relation using Boolean matrix multiplication.

These properties help analyze structures in set theory, graph theory, and databases.

Pseudocode/Flowchart:

Main Function:

1. Start loop (`keepGoing == 'y'`).
2. Input the size of the set `sizeOfA`.
3. Input elements of the set `setA[]`.
4. Input the number of pairs in the relation `sizeOfB`.
5. Input the relation pairs in `relB[]` as a 2D array.
6. Call the following functions:
    - `reflexive(setA, sizeOfA, relB, sizeOfB)`
    - `symmetric(setA, sizeOfA, relB, sizeOfB)`
    - `antiSymmetric(setA, sizeOfA, relB, sizeOfB)`
    - `transitive(setA, sizeOfA, relB, sizeOfB)`
7. Ask the user if they want to run the program again (`keepGoing`).
8. End loop when `keepGoing == 'n'`.

Function: pair_is_in_relation(e1, e2, relB[], sizeOfB)

1. For each pair in the relation array `relB[]`:
    - Check if the pair `(e1, e2)` exists.
    - Return `true` if found, otherwise `false`.

Function: reflexive(setA[], sizeOfA, relB[], sizeOfB)

1. For each element `a[i]` in `setA[]`, check if the pair `(a[i], a[i])` exists in `relB[]`:
    - If any pair does not exist, print "Reflexive - No" and return `false`.
2. If all pairs are found, print "Reflexive - Yes" and return `true`.

Function: symmetric(setA[], sizeOfA, relB[], sizeOfB)

1. For each pair `(e, f)` in `relB[]`, check if the pair `(f, e)` also exists:
   - If not, print "Symmetric - No" and return `false`.
2. If all pairs are symmetric, print "Symmetric - Yes" and return `true`.

Function: antiSymmetric(setA[], sizeOfA, relB[], sizeOfB)

1. For each pair `(e, f)` in `relB[]`, if `e != f` and the pair `(f, e)` also exists:
   - Print "AntiSymmetric - No" and return `false`.
2. If all conditions are satisfied, print "AntiSymmetric - Yes" and return `true`.

Function: transitive(setA[], sizeOfA, relB[], sizeOfB)

1. For each pair `(e, f)` in `relB[]`, find pairs where `f == b[j]`:
   - Check if the pair `(e, g)` exists where `g = b[j+1]`.
   - If any such pair is missing, print "Transitive - No" and return `false`.
2. If all transitive conditions are met, print "Transitive - Yes" and return `true`.

Implementation:

```
#include <bits/stdc++.h>

using namespace std;


bool pair_is_in_relation(int e1, int e2, int b[], int sizeOfB)

{

    for (int i = 0; i < sizeOfB; i += 2)

    {

        if (b[i] == e1 && b[i+1] == e2)

            return true;

    }

    return false;

}


bool reflexive(int a[], int sizeOfA, int b[], int sizeOfB)

{

    for (int i = 0; i < sizeOfA; i++)

    {

        if (!pair_is_in_relation(a[i], a[i], b, sizeOfB))

            return false;

    }

    cout << "Reflexive - Yes" << endl;

    return true;

}
```

```
bool symmetric(int a[], int sizeOfA, int b[], int sizeOfB)

{

    for (int i = 0; i < sizeOfB; i += 2)

    {

        int e = b[i];

        int f = b[i+1];

        if (!pair_is_in_relation(f, e, b, sizeOfB))

            return false;

    }

    cout << "Symmetric - Yes" << endl;

    return true;

}


bool antiSymmetric(int a[], int sizeOfA, int b[], int sizeOfB)

{

    for (int i = 0; i < sizeOfB; i += 2)

    {

        int e = b[i];

        int f = b[i+1];

        if (e != f && pair_is_in_relation(f, e, b, sizeOfB))

            return false;

    }

    cout << "AntiSymmetric - Yes" << endl;
```

```
    return true;

}


bool transitive(int a[], int sizeOfA, int b[], int sizeOfB)

{

    for (int i = 0; i < sizeOfB; i += 2)

    {

        int e = b[i];

        int f = b[i+1];

        for (int j = 0; j < sizeOfB; j += 2)

        {

            if (f == b[j])

            {

                int g = b[j+1];

                if (!pair_is_in_relation(e, g, b, sizeOfB))

                    return false;

            }

        }

    }

    cout << "Transitive - Yes" << endl;

    return true;

}


int main()
```

```
{
    char keepGoing = 'y';
    while (keepGoing == 'y')
    {
        int sizeOfA;
        cout << "Enter the size of the set: ";
        cin >> sizeOfA;

        int setA[sizeOfA];
        cout << "Enter the elements of the set: ";
        for (int i = 0; i < sizeOfA; i++)
        {
            cin >> setA[i];
        }

        int sizeOfB;
        cout << "Enter the number of pairs in the relation: ";
        cin >> sizeOfB;
        sizeOfB *= 2;

        int relB[sizeOfB];
        cout << "Enter the relation pairs (as pairs of integers):" <<
endl;
        for (int i = 0; i < sizeOfB; i += 2)
```

```
        {
            cout << "Pair " << (i / 2) + 1 << ": ";
            cin >> relB[i] >> relB[i + 1];
        }

        cout << "Results for the entered set and relation:" << endl;
        reflexive(setA, sizeOfA, relB, sizeOfB);
        symmetric(setA, sizeOfA, relB, sizeOfB);
        antiSymmetric(setA, sizeOfA, relB, sizeOfB);
        transitive(setA, sizeOfA, relB, sizeOfB);

        cout << endl << "Would you like to test it again? (y/n): ";
        cin >> keepGoing;
    }

    return 0;
}
```

Output:

```
d:\code\Pro\cpp>cd "d:\code\Pro\cpp\" && g++ tempCodeRunnerFile.cpp -o tempCodeRunnerFile && "d:\code\Pro\cpp\"tempCodeRunnerFile
Enter the size of the set: 3
Enter the elements of the set: 1 2 3
Enter the number of pairs in the relation: 4
Enter the relation pairs (as pairs of integers):
Pair 1: 1 1
Pair 2: 2 2
Pair 3: 1 2
Pair 4: 3 1
Results for the entered set and relation:
AntiSymmetric - Yes

Would you like to test it again? (y/n): y
Enter the size of the set: 3
Enter the elements of the set: 1 2 3
Enter the number of pairs in the relation: 4
Enter the relation pairs (as pairs of integers):
Pair 1: 1 1
Pair 2: 2 2
Pair 3: 1 2
Pair 4: 2 1
Results for the entered set and relation:
Symmetric - Yes
Transitive - Yes

Would you like to test it again? (y/n):
```

Result/Discussion:

The program successfully tested a set and its relation for four properties: reflexive, symmetric, anti-symmetric, and transitive.

- Reflexive: The relation was not reflexive because some elements did not relate to themselves (e.g., (3, 3) was missing).
- Symmetric: The relation was symmetric, as for every pair (a, b), the reverse pair (b, a) existed.
- Anti-Symmetric: The relation was not anti-symmetric, since both (1, 2) and (2, 1) existed, violating the anti-symmetry condition for $a \neq b$.
- Transitive: The relation was transitive, meaning indirect connections between elements were correctly represented.

In summary, the relation in the example passed the symmetry and transitivity checks but failed the reflexivity and anti-symmetry tests.

Applications:

Database Management: Reflexive, symmetric, and transitive properties are essential in ensuring data integrity and optimizing query processing, especially in relational databases for operations like join and closure.

Graph Theory: These properties are used to analyze the structure of graphs, where nodes represent entities and edges represent relationships. Transitive closure, for example, is crucial for finding reachability in networks.

Computer Networks: Reflexive and transitive relations are vital in determining network reachability, while symmetry helps in understanding bidirectional connections between nodes.

Mathematical Logic: Set relations and their properties are fundamental in fields like set theory, algebra, and logic, particularly in defining equivalence relations and partial orders.

Social Networks: In social network analysis, symmetric relations reflect mutual connections, while transitivity models influence, suggesting how relationships might propagate through indirect connections.

References/Research Papers: (In IEEE format)

S. Warshall, "A Theorem on Boolean Matrices," *Journal of the ACM (JACM)*, vol. 9, no. 1, pp. 11–12, 1962, doi: 10.1145/321105.321107.

R. E. Tarjan, "Depth-First Search and Linear Graph Algorithms," *SIAM Journal on Computing*, vol. 1, no. 2, pp. 146-160, 1972, doi: 10.1137/0201010.

A. Silberschatz, H. F. Korth, and S. Sudarshan, *Database System Concepts*, 6th ed., New York, NY, USA: McGraw-Hill, 2010.

T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms*, 3rd ed., Cambridge, MA, USA: MIT Press, 2009.

G. Gallo, G. Longo, S. Pallottino, and S. Nguyen, "Directed Hypergraphs and Applications," *Discrete Applied Mathematics*, vol. 42, no. 2, pp. 177–201, 1993, doi: 10.1016/0166-218X(93)90035-3.

K. S. Rosen, *Discrete Mathematics and Its Applications*, 7th ed., New York, NY, USA: McGraw-Hill, 2012.