Q3)

#include <iostream>

using namespace std;

class Relation

{

int n;

int \*\*arr;

public:

Relation(int s)

{

n = s;

arr = new int \*[n];

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

arr[i] = new int[n];

}

void takeInput()

{

cout << "Enter Matrix elements" << endl;

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

{

for (int j = 0; j < n; j++)

cin >> arr[i][j];

}

}

void fromArray(int \*\*a)

{

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

{

for (int j = 0; j < n; j++)

arr[i][j] = a[i][j];

}

}

void printRelation()

{

cout << " ";

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

cout << (char)('a' + i) << " ";

cout << endl;

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

{

cout << (char)('a' + i) << " ";

for (int j = 0; j < n; j++)

cout << arr[i][j] << " ";

cout << endl;

}

}

bool isReflexive()

{

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

{

if (!arr[i][i])

return false;

}

return true;

}

bool isSymmetric()

{

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

{

for (int j = i; j < n; j++)

{

if (arr[i][j] != arr[j][i])

return false;

}

}

return true;

}

bool isAntiSymmetric()

{

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

{

for (int j = i + 1; j < n; j++)

{

if (arr[i][j] == arr[j][i] && arr[i][j]) // ignore where arr[i,j] = arr[j,i] = 0

return false;

}

}

return true;

}

bool isTransitive()

{

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

{

for (int j = 0; j < n; j++)

{

// Check if i,j = 1 go for j,k to check for 1 then i,k

if (arr[i][j])

{

for (int k = j; k < n; k++)

{

if (arr[j][k] && !arr[i][k])

return false;

}

}

}

}

return true;

}

bool isPartiallyOrdered()

{

return isReflexive() && isAntiSymmetric() && isTransitive();

}

bool isEquivalence()

{

Q$) return isReflexive() && isSymmetric() && isTransitive();

}

};

int main()

{

Relation R(3);

R.takeInput();

cout << "Relation - Adjacency matrix" << endl;

R.printRelation();

cout << "Reflexive: " << R.isReflexive() << endl;

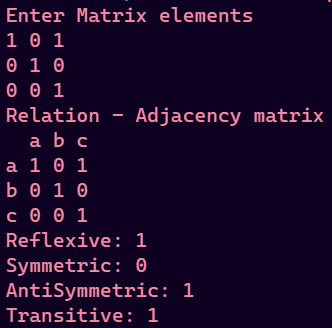
cout << "Symmetric: " << R.isSymmetric() << endl;

cout << "AntiSymmetric: " << R.isAntiSymmetric() << endl;

cout << "Transitive: " << R.isTransitive() << endl;

return 0;

}



Q4)

#include <iostream>

using namespace std;

class Relation

{

int n;

int \*\*arr;

public:

Relation(int s)

{

n = s;

arr = new int \*[n];

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

arr[i] = new int[n];

}

void takeInput()

{

cout << "Enter Matrix elements" << endl;

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

{

for (int j = 0; j < n; j++)

cin >> arr[i][j];

}

}

void fromArray(int \*\*a)

{

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

{

for (int j = 0; j < n; j++)

arr[i][j] = a[i][j];

}

}

void printRelation()

{

cout << " ";

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

cout << (char)('a' + i) << " ";

cout << endl;

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

{

cout << (char)('a' + i) << " ";

for (int j = 0; j < n; j++)

cout << arr[i][j] << " ";

cout << endl;

}

}

bool isReflexive()

{

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

{

if (!arr[i][i])

return false;

}

return true;

}

bool isSymmetric()

{

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

{

for (int j = i; j < n; j++)

{

if (arr[i][j] != arr[j][i])

return false;

}

}

return true;

}

bool isAntiSymmetric()

{

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

{

for (int j = i + 1; j < n; j++)

{

if (arr[i][j] == arr[j][i] && arr[i][j]) // ignore where arr[i,j] = arr[j,i] = 0

return false;

}

}

return true;

}

bool isTransitive()

{

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

{

for (int j = 0; j < n; j++)

{

// Check if i,j = 1 go for j,k to check for 1 then i,k

if (arr[i][j])

{

for (int k = j; k < n; k++)

{

if (arr[j][k] && !arr[i][k])

return false;

}

}

}

}

return true;

}

bool isPartiallyOrdered()

{

return isReflexive() && isAntiSymmetric() && isTransitive();

}

bool isEquivalence()

{

return isReflexive() && isSymmetric() && isTransitive();

}

};

int main()

{

Relation R(3);

R.takeInput();

cout << "Relation - Adjacency matrix" << endl;

R.printRelation();

cout << "Partially Ordered: " << R.isPartiallyOrdered() << endl;

cout << "Equivalence: " << R.isEquivalence() << endl;

return 0;

}

