CISP 440

Austin Smothers

Homework 6

/\*

Reads in a matrix from a binary file and determines RST and EC's.

Austin Smothers Feb 2017

\*\*Based loosely on code provided by Professor Dan Ross\*\*

#include <iostream>

#include <string.h>

#include <string> //I need string to use a string as a variable name later

#include <fstream>

#pragma warning( disable : 4267)

#define MAX 40

using namespace std;

/\* The Matrix. No-one can be told what the Matrix is, you must be shown.

#define BLUEPILL

#define REDPILL

\*/

int R[MAX][MAX]; // a boolean array indicating members of a relation

int R2[MAX][MAX]; // a boolean array meant to store R squared

int EC[MAX]; // a boolean array indicating representatives of equivalence classes

int size;

/\*

This prints a matrix

\*/

void printMatrix(int R[MAX][MAX])

{

int i, j;

/\*

using size doesn't work here. It's a global variable modified in main

but initialized to 0, and thus this function views the value of size

as 0 (or worse, undeclared! Instead I prefer to just use sizeof()

to get the size of the array to be printed, so I did just that

\*/

for(i = 0; i < sizeof(R); i++)

{

for(j = 0; j < sizeof(R); j++)

cout << R[i][j];

cout << endl;

}

}

/\*

Checks if a matrix is reflexive.

Checks the diagonals of a matrix. All elements on the diagonal

must be 1.

\*/

bool IsRefx(int R[MAX][MAX])

{

int i = 0; //i simply functions as a for loop counter

//a test function to make sure that size-of works the way I want it

//to with 2-d arrays, which it does!

//cout << sizeof(R) << endl;

//standard double for loop set-up for a 2-d array

for (i; i < sizeof(R); i++)

{

if (R[i][i] == 0) //checks the diagonal for 1's. If any

{ //member isn't a 1, then !Refx

cout << "This array"

" is not reflexive!\n";

return false; //empty return statement to prevent

//output of the below text

}

}

cout << "This array is Reflexive!\n";

return true;

}

/\*

Checks is a matrix is symmetric

i,j == j,i for every cell of the matrix

\*/

bool IsSymt(int R[MAX][MAX])

{

int i, j = 0;

//standard double for loop set-up for a 2-D array

for (i = 0; i < sizeof(R); i++)

{

for (j = 0; j < sizeof(R); j++)

{

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

{ //if there is any exception to symmetry, the function returns

//false, proving that the entire array is not symmetric

cout << "This array is not Symmetric!\n";

return false; //return statement to prevent

//output of the below text

}

}

}

//if the array makes it through every iteration of the for loops,

//then it is symmetric

cout << "This array is Symmetric!\n";

return true;

}

/\*

Checks if a matrix is transitive.

First calculate the matrix squared

Every non-zero element of Rsquared must also be non zero in R

\*/

bool IsTrans(int R[MAX][MAX], int R2[MAX][MAX])

{

int i, j = 0;

//typical double for loop set-up to run through all elements of the arrays

for (i = 0; i < sizeof(R); i++)

{

for (j = 0; j < sizeof(R); j++)

{

//if statement checking to ensure that if R2 has a nonzero element

//in one position and R has a zero element in that position, then

//R must not be transitive according to the check

if ((R2[i][j] != 0) && (R[i][j] == 0))

{

cout << "This array is not Transitive!\n";

return false; //return statement to prevent further execution

//of function

}

}

}

//if the arrays R and R2 have made it all the way through the for loops

//then R must be transitive. Let's share the good news!

cout << "This array is Transitive!\n";

return true;

}

/\*

Squares a matrix

R2 = R x R

R2 i,j = sum of R i,k \* R k,j

\*/

bool SquareMatrix(int R[MAX][MAX], int R2[MAX][MAX])

{

//used to store whether IsTrans returns true or false

bool tf = true;

int i, k, j, sum = 0;

/\*this loop will require three for loops, as

Rsquared = R[i][k] \* R[k][j]

incremented in order of j, k, i \*/

for (i = 0; i < sizeof(R); i++)

{

for (j = 0; j < sizeof(R); j++)

{

for (k = 0; k < sizeof(R); k++)

{

sum += (R[i][k] \* R[k][j]);

//this inner loop obtains the sum of the array

//elements to be multiplied

}

//sets R2 to the sum of the previously multiplied elements

R2[i][j] = sum;

//resets sum for the next array operation

sum = 0;

}

}

//check to make sure I coded this properly

cout << "Current Matrix Squared:\n";

printMatrix(R2);

//calls IsTrans to determine if R is Transitive

tf = IsTrans(R, R2);

return tf;

}

/\*

Iterate thru the captains array. For each captain, go to that row of

the matrix and print the members of the class.

\*/

void printECs(int R[MAX][MAX], int EC[MAX])

{

int i, j = 0;

//this chunk prints line 0

cout << "|" << EC[0] << "| : {";

for (j = 0; j < sizeof(R); j++)

{

if (R[0][j] == 1)

{

cout << " " << j;

}

}

cout << " }\n";

//this chunk checks if a row has a captain, and prints the line if it does

for (i = 0; i < sizeof(EC); i++)

{

if (EC[i])

{

cout << "|" << EC[i] << "| : {";

//for a given captain, print out each member of R

//whose value is set to 1

for (j = 0; j < sizeof(R); j++)

{

if (R[EC[i]][j] == 1)

{

cout << " " << j;

}

}

cout << " }\n";

}

}

}

/\*

Finds equivalence classes and elects representatives of each class.

\*/

void FindECs(int R[MAX][MAX], int EC[MAX])

{

int i, j, k = 0;

EC[0] = 0;

if (IsRefx(R) && IsSymt(R) && SquareMatrix(R, R2))

{

cout << "This array is an Equivalence Relation!\n"

"Equivalence Classes: \n";

//standard for loop set-up for a 2-d array

for (i = 0; i < sizeof(R); i++)

{

for (j = 0; j < sizeof(R); j++)

{

for (k = 3; k > -1; k--)

{ //check each row for any elements similar to previous rows

//but also make sure that a row cannot be checked against

//itself

if ((R[k][j] == R[i][j]) && (k != i))

i++;

//ensures that each row's elements are checked against all

//other elements in the same column

else if (k != 0);

//if a row contains no common elements with any other

//rows then make its row # a captain

else

EC[i] = i;

}

}

}

printECs(R, EC);

}

else

cout << "This array does not generate an Equivalence Class.\n";

}

void main(void)

{

char c;

// declare source file

ifstream fin;

/\*

The following code allows me to use a string to represent file names.

This allows me to use a for loop to run thru all 7 binary files in one run

and also conveniently allows me to output which file the program is iterating

on

\*/

string filename; //this is the string that stores the file name

for (int n = 1; n < 8; n++)

{

//this switch statement uses n and switches to determine what file's name

//to set filename equal to

switch (n)

{

case 1: filename = "R1.bin";

break;

case 2: filename = "R2.bin";

break;

case 3: filename = "R3.bin";

break;

case 4: filename = "R4.bin";

break;

case 5: filename = "R5.bin";

break;

case 6: filename = "R6.bin";

break;

case 7: filename = "R7.bin";

break;

}

//open source file, replacing the text with a reference to filename

//determined by the above switch statement

fin.open(filename.c\_str(), ios\_base::binary);

if (!fin) { cerr << "Input file could not be opened\n"; exit(1); }

// get the matrix size

fin.read(&c, 1); size = c;

// fill the matrix from the file

int i, j;

for (i = 0; i < size; i++)

for (j = 0; j < size; j++)

{

fin.read(&c, 1);

R[i][j] = c;

}

// close file

fin.close();

//print the file name

cout << endl << filename << endl;

//prints the matrix in question

printMatrix(R);

//this function contains a true/false check for Refx, Symt, and Trans

//therefore, calling FindECs will run all of the aforementioned functions

FindECs(R, EC);

}

//The program is polite and says goodbye

cout << "Job's Done!\nGoodbye!\n";

}

