COSC 3319 Data Structures Spring 2018

Due: Thursday February 15. THIS LAB MAY NOT BE SUBMITTED FOR CREDIT AFTER THE DUE DATE! Expect additional labs to be assigned during this time frame. You may use Windows, UNIX (any flavor including Apple) or Chrome operating system. Other operating systems will be considered on request.

This lab is meant to push you beyond your current level of expertise (comfort) to new planes of achievement, i.e., professional grade. It purposely contains elements that will require you to research new solution techniques. A goal of the assignment is provide you the opportunity to research and master new technology either on your own or with a minimum of external help. I am always available for questions. ***Labs must be submitted in an envelope sufficiently large to hold your results without having to fold the lab materials. Write your name, "COSC 3319," section number and time the class meets in the upper left hand corner of the envelope opposite the side the flap is folded***. You may use the same envelope for all labs during the semester.

You must clearly state on the first page of your program which grading option you have successfully completed. Programs failing to state the grading option will be graded as "D" option programs. Submit a printed copy of all code, any data files, and results. Do not embarrass yourself or me with screen shots. This lab is not technically challenging with respect to data structures. Its primary purpose is to ensure you gain the required background to complete "professional" work on future labs.

Additional labs will be assigned prior to the due date. Do not procrastinate finishing this first assignment. It will help you learn the new required language technology to implement the advanced data structures labs. This lab is designed so you can start by implementing the "C" option. The "C" option can be converted to the "B" option, leading to the "A" option with minimal loss of effort. **In essence. You tell me your professional worth!** Be "Professional Grade!" Submit "A+" Work.

Most of the grading options give you a choice of languages. You are encouraged to use Ada for several reasons. First, if you ever work with a company contracting with DOD, (Department of Defense), Aero Space industry or European industries, knowledge of Ada is an advantage. You should be able to use Ada, C++ or C# for the "B" and "A" options on all labs. You may use Ada, C++, C#, Java or assembly for the "C" and "B" options on this lab. You *may not use* Perl, Groovy, Ruby and Python. They do not support the abstractions required for real time environments and the advanced grading requirements on most labs. They are interpretive languages designed for other application areas including commercial web based environments. We will emphasize compiled languages this semester oriented to real time applications with some latitude. Interpretive languages tend to be 10 to 30 times slower than compiled languages even with JIT (just in time translation). Substantial examples using Ada will be utilized in class over the next few weeks

If you work for bleeding edge technology companies, you will have to learn to change programming languages like you change your clothes. The more languages you know prior to graduation, the easier it will be to learn new languages on the job after

graduation. Most companies will expect you to learn new languages on your on time using your own resources.

PROBLEM SPECIFICATION:
Several major companies needing help in solving a network communications problem have approached us. Essentially, the desire is given partial information about network connectivity in the form of a network diagram, we have been asked to develop a method, which indicates all nodes reachable on the network from any node specified with the available information. Normally we will not really have the network diagram but rather a set of relations in the form (node[J] , node[K]) indicating communications currently exist from node J to node K. Some network nodes are not really destinations in the traditional sense, but special nodes (repeaters) used to facilitate communications between nodes separated by a large distance. These nodes receive network traffic, determine network routing, and retransmit the packet towards the desired destination.

To support high traffic volume, two separate half duplex communication lines too neighboring nodes normally connect each node in our network. One line is used to receive incoming traffic (packets) from the connected node; the other line is used to transmit outgoing packets. It is possible for one or both of these communications lines to be down at any given time. Hence it is possible for a network node to be able to receive but not transmit and vice versa. In the following sample network diagram, the arrows indicate the directions in which network traffic may currently travel between nodes.

Test your code using the following sample network diagram. Rows and columns must be properly labeled using the node names in the presentation of results. Node names may be kept as characters, strings or as an enumeration type. Fixed and variable length strings may also be used to represent row and column labels if desired.

We will be use Warshall's Algorithm to generate the transitive closure of a set of relations. This algorithm is at the heart of most programming language translators. A programming language translator must accept an infinite number of legal programs and reject an infinite number of illegal programs. This algorithm makes it possible to accomplish this task and generate error messages when required.

Data for the "C" Option:

Input relations: First(A,B), First(B,D), First(C,B), First(B,C), First(J,K), First((J,L)

Using "1" for true and "0" for false (false entries were left blank to improve readability) the above relations would appear as follows. This is frequently referred to as a Boolean Matrix Representation (BMR) and represents all reachable destinations from a node (row) to other nodes (columns) in a single step ($First^1$).

	A	B	C	D	J	K	L
A		1					
B			1	1			
C		1					
D							
J						1	1
K							
L							

This representation is frequently referred to as a Boolean Matrix Representation (BMR) and represents all reachable destinations from a given node (row) to other nodes (columns) in a single step ($First^1$). What we desire is set of all nodes reachable from the given node (row), the transitive closure of the relation First usually denoted as $First^+$.

First + for the above communications network after applying Warshall's algorithm follows:

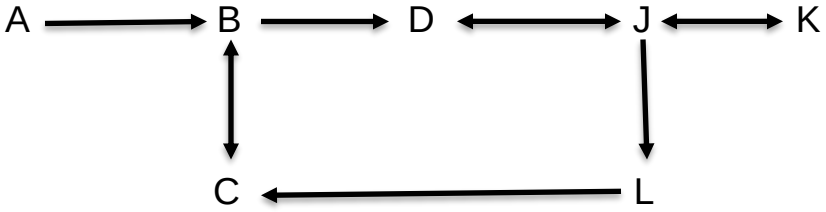
	A	B	C	D	J	K	L
A		1	1	1			
B		1	1	1			
C		1	1	1			
D							
J						1	1
K							
L							

As an example, node A can communicate with nodes B, C and D. Node J can only communicate with nodes K and L.

Using Boolean variables (an intrinsic data type, BOOLEAN is (FALSE, TRUE)) the initial matrix above would appear as follows. The "C" data set would appear as follows when represented as a BOOLEAN matrix. Ada supports "not," "and," "or" and "xor" for BOOLEAN variables.

	A	B	C	D	J	K	L
A	FALSE	TRUE	FALSE	FALSE	FALSE	FALSE	FALSE
B	FALSE	FALSE	TRUE	TRUE	FALSE	FALSE	FALSE
C	FALSE	TRUE	FALSE	FALSE	FALSE	FALSE	FALSE
D	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE
J	FALSE	FALSE	FALSE	FALSE	FALSE	TRUE	TRUE
K	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE
L	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE

Data for the "B" Option:

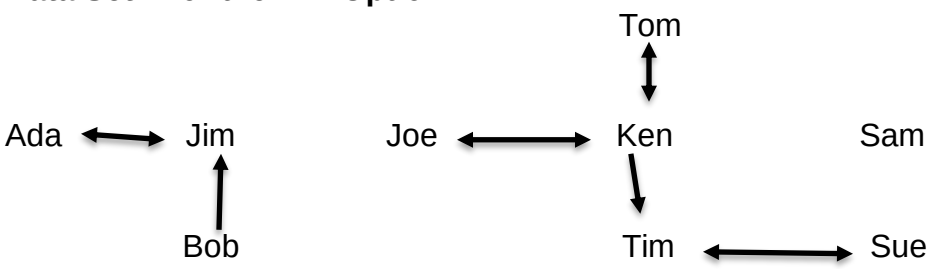


Input relations: First(A,B), First(B,D), First(C,B), First(B,C), First(J,D), First(D,J) First(J,L), First (L,C), First(J,K), First(K, J)

The results should appear in a matrix labeled as follows:

	A	B	C	D	J	K	L
A		1					
B			1	1			
C		1					
D					1		
J				1		1	1
K					1		
L			1				

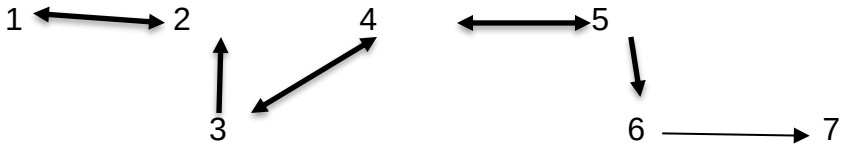
Data Set 1 for the "A" Option:



The results should appear in a matrix labeled as follows:

	Ada	Bob	Joe	Ken	Sam	Sue	Tim	Tom	Jim
Ada									1
Bob									1
Joe				1					
Ken			1				1	1	
Sam									
Sue							1		
Tim						1			
Tom				1					
Jim	1								

Data Set 2 for the "A" Option:



The results should appear in a matrix labeled as follows using 0 for false and 1 for true:

	1	2	3	4	5	6	7
1							
2							
3							
4							
5							
6							
7							

"C" Option (best grade is 75).

You may use Ada, C++, C#, Java. If you desire to use another language it must be supported by IT at Sam or on the CS network and approved prior to use by the course instructor on an individual basis. You are encouraged to use Ada, function/procedures properly but may write the program as monolithic code.

The data for the lab must be entered dynamically at runtime as Boolean values and the results written to a file as Boolean values. You are encouraged to place the input data in a file and use "I/O redirection" (see hints) for the "C" option. You may use "I/O redirection" to store the results in a file. I/O redirection is a technique that should be mastered by all professionals both programmers and users. The results should be printed as type Boolean.

Process the information in the "C" Option sample network diagram using Warshall's Algorithm to compute the transitive closure. **Prompt the user for the size of the array to hold the BMR (Boolean Matrix Representation, number of rows and columns) at runtime and dynamically allocate the array in the system stack** in Ada, C++, or Java. *You specifically should not use "new, malloc," or any other operator, which allocates space in the heap in any language. Clearly mark this section of your code with a highlighter!* Print each transaction as it is processed. I recommend the use of a data file. It will save you time and effort. *With great reluctance, I will not take off for storing the data structure in the heap on the "C" option*.

Submission: Submit a cover page with your name, COSC3319.xx where "xx" is the section number, time the class meets and date of submission. Clearly indicate you are submitting the "C" grading option. The cover page should be followed by the properly labeled "C" option results then the "C" option code.

"B" Option (best grade is 85):

First complete the "C" Option using Boolean values. Second, implement Warshall's algorithm as an Ada package (class for C++/C#/Java). Build the BMR in your main program using the integers 0 for false and 1 for true. Pass the BMR as a parameter to the package/class method implementing Warshall's algorithm. *You specifically may not use "new, malloc," or any other operator, which allocates space in the heap in any language. Clearly mark the section of your code where you allocate the space with a highlighter!* **A 10 point penalty will be charged on your grade if you allocate storage in heap memory anywhere in your program**. The BMR should be returned to the main program by reference (CBR). Print the results of Warshall's algorithm in the main program using 0 and 1 for false and true**. All I/O must occur in the main program.** You may assume the names of network nodes are a single character.

You may use I/O redirection for files to complete the "B" Option.

Submission: Submit a cover page with your name, COSC3319.xx where "xx" is the section number, time the class meets and date of submission. Clearly indicate you are submitting the "B" grading option. The cover page should be followed by the "C" option results followed by the "C" option code. The "B" Option results should appear next followed by the "B" option code.

Hint: BMR(I,J) or BMR(K,L): if BMR(I,J) + BMR(K,L) = 1 or greater than 1, the result is 1. If the result of BMR(I,J) + BMR(K,L) = 0 the result is 0.

Hint: Over the years I have used calculating the transitive closure of a relation many times as one of the problems in a set for ACM and high school student programming contest. High school students usually solve the problem as follows when 0 represents false and 1 represents true. There are several other ways to accomplish this task. CS is about problem solving, thinking out of the box, not memorized solutions.

Assume 2 integer variables "A" and "B" containing either a 0 or 1 (False or True) where "+" is treated as the Boolean "or" operation.
If (A + B) = 0 the result is false. – if (A or B) = 0 the desired result is 0.
If (A + B) >= 1 the result is true. – If(A or B) >= 1 the desired result is 1.

"A" Option (best grade is 100):

You need not explicitly write the code for the "C" and "B" options. Please use generic instantiations of packages/classes. Space for each BMR must be allocated in the system stack within the generic package/template, probably during generic instantiation! *You specifically may not use "new," "malloc," or any other operator, which allocates space in the heap at runtime in any language. Clearly mark this section of your code with a highlighter!* **You must read all transactions and print all results within the generic package/template. The I/O routines must be passed as generic parameters (Ada) or as pointers to functions (C++/C#/Java). You will be expected to pass an overload for the "or" operator or function to the generic package/template. Upon return to the main program, all storage required for the BMR must be reclaimed efficiently and automatically by the OS without programmer intervention or use of a garbage collector. The array subscripts should be an enumeration type, i.e., the labels used for the rows and columns. This implies the enumeration type should be passed as a generic/template parameter. All data must be read explicitly from a sequential text file and results printed to a text file *in the generic package/class*!. *You may not use I/O redirection*.** Use a highlighter to mark where you open and close the text file.

Ada encourages programmers to define and use their own data types. C++ and Java restricts programmers to intrinsic data types in generics /templates without the use of pointers. C++ and Java do support enumeration type (enum) I/O. You will need to write your own I/O routines to print the results. Ada allows convenient instantiation and I/O operations with any enumeration or programmer defined data type. You must use Ada, C++ or C# for the "A" option.

Submission: Submit a cover page with your name, COSC3319.xx where "xx" is the section number, time the class meets and date of submission. Clearly indicate you are submitting the "A" grading option. The cover page should be followed by the "C" option results followed by the "B" Option results then the "A" Option results. The "A" Option code should appear next.

Process all data files in a single execution of your program making sure you return the space occupied by previous BMR's to the OS prior to processing the next BMR.

Extensions for the use of this lab might include:

1) **We must bring down a node for P.M. (preventive maintenance). How will the network be effected?**
2) **We wish to improve network reliability. How will inserting a node at a given point influence connectivity?**

HINTS:

Hint 1: Assume the name of your program is Pgm1.exe. If you type "Pgm1" at the DOS prompt, input is normally expected from the keyboard and results are printed on the terminal. The command "Pgm1 > file1" would expect all input to come from the keyboard but the results would be routed to a disk file in the current directory named "file1." The command "Pgm1 < raw1 > results" would obtain input (as stream IO) from the file "raw1" and place the output in the text file "results" in the current directory. You may include path names for files, e.g., D:\\home\\sub1\\raw1). Most operating systems interpret a "\" or "/" as the escape character. Two slashes in a row are interpreted as a single slash.

Hint 2:
Normally Ada will not allow an integer to be assigned to a character variable as bits must be truncated. Similarly, Ada will not normally allow a character to be assigned to an integer as there are not enough bits (padding must occur). In both cases, Ada will flag a probable error at compile time letting the programmer know they have probably made a logic error. The same is true if you try to do arithmetic operation on different data types such as adding an integer to a floating point number as data conversions must occur. In general this is a poor programming practice as storage types differ forcing data conversions. The conversions add significantly to the runtime of programs.

To tell the compiler you really mean to perform the indicated operation, the generic package "Unchecked_Conversion" should be instantiated to allow the conversion as shown below. There is no actual runtime penalty for most instantiations of Unchecked_Conversion (no run time function call) resulting in CPU overhead when using a good Ada compiler. The generic conversion is simply allowed at compile time as in the example below.

```
with Ada.Text_IO; use Ada.Text_IO;  -- read and write characters.
with Unchecked_Conversion;  -- standard package with every validated Ada
translator

procedure ConvertCharacterInteger is
    -- To read and write 16 bit integers on the PC compiler.
    package MyInt_IO is new Ada.Text_IO.Integer_IO(integer);
    use MyInt_IO;
```

```
    -- instantiations to convert between integer and character formats, 16
bits versus 8 bits.
    function integerToCharacter is new Unchecked_Conversion(Integer,
Character);
    function characterToInteger is new Unchecked_Conversion(Character,
Integer);

    c1, c2: Character;
    int1, int2: Integer;

    begin
       c1 := 'A';
       -- int1 := c1;  --error in Ada, strongly typed, suspects programmer
error.
       int1 := characterToInteger( c1 );  -- Signal compiler to allow
conversion.
       put(" int1 = "); put(int1,4); put("  ");
       put("c1 = "); put(c1); new_line(2);

       int2 := 66;
       -- c2 := int2;  -- error
       c2 := integerToCharacter(int2);
       put(" int2 = "); put(int2,4); put("  ");
       put("c2 = "); put(c2); new_line(2);


end ConvertCharacterInteger;
```

In application programming, it is occasionally desirable to treat the same unit of memory at different times as a different data type!

Frequently used conversion trick in assembly, ‘‘C,’’ occasionally Ada and other languages.

ASCII Conversions

Character	Decimal	Integer	Integer (32 bits) Binary
‘0’	48	0	0-0000
‘1’	49	1	0-0001
‘2’	50	2	0-0001
‘3’	51	3	0-0011
‘4’	52	4	0-0100
‘5’	53	5	0-0101
‘6’	54	6	0-0110
‘7’	55	7	0-0111
‘8’	56	8	0-1000
‘9’	57	9	0-1001

Assume we wish to convert a 32 bit one digit integer to an 8 bit ASCII character. This may be accomplished by adding the character ‘0’ (48 decimal) to the integer and dropping the leading 24 zeros. As an example the 32 bit integer 3 may be converted to a character by adding the character ‘0’ (or decimal 48) then truncating the leading 24 bits.

Ex: 3 + 48 = 3 + ‘0’ = 51 => ’3’ in ASCII

Alternately an ASCII digit represented as character may be converted to an integer by subtracting the character '0' (or 48) and padding the 24 bits to the left with zeros.

Ex: '3' – 48 = 51 – 48 = 3 => 3 in base 10 or 11 in binary.

Using "C/C++/C#"
ch: char = '3';
int1: int = 0;

int1 = (int)ch - 48; //first coerce/cast the character to an integer then convert to integer 3.
Or alternately:
int1 = int(ch) – 48; //function form of casting.

ch = char(3 + 48); // yields the character 3.
Or
ch = (char) (3 + 48); // yields the character 3.

In COBOL, use the "redefines verb."

Hint 3:
Assume (Ada):

```
type MonthName is (January, February, March, April, May, June, July, August,
                   September, October, November, December);
-- have compiler write I/O routines to print programmer defined enumeration type.
package MonthNameIO is new Ada.Text_IO.Enumeration_IO(MonthName);
use MonthNameIO;
-- I/O routines to read and write integers.
Package IntegerIO is new Ada.Text_IO.Enumeration_IO(Integer);
Use IntegerIO;

type DateType is record
   month: MonthName;  day: integer range 1..31;  year: integer;
end record;

aDate: DateType := (January, 15, 1947); -- A typical declaration and assignment.

procedure printDate( adate: in DateType) is
begin
        put(adate.month);  put(adate.day,3); put(" "); put(adate.year); new_line;
end printDate;
```

The "3" in the above print statement indicates the day is to be printed right justified using 3 spaces.

Hint 4:

```
--in file:  Using_Text_Files.adb
with Ada.Text_IO;               use Ada.Text_IO;
with Unchecked_Conversion;
```

```
procedure Using_Text_Files is
   Input, Output : File_Type;  -- Logical file names.  File_Type data
structure from Text_IO.

   char:    Character;
   question:  Boolean;
   bool1, bool2:  Boolean;

   lf_integer: Integer := 8#012#; -- Special effects, ASCII value for linefeed
(lf) in octal.
   lf:      Character;
   -- function to convert integer to character for linefeed on output.
   function integerToCharacter is new Unchecked_Conversion(Integer,
Character);

   type empNames is (Joe, Sam, Tom);  -- Declare enumeration type and write
I/O routines.
   package empNames_IO is new Ada.Text_IO.Enumeration_IO(empNames);
   use empNames_IO;
   nameIN:  empNames;

   package Boolean_IO is new Ada.Text_IO.Enumeration_IO(Boolean);  -- I/O for
Boolean.
   use Boolean_IO;

   package Integer_IO is new Ada.Text_IO.Enumeration_IO(Integer);  -- I/O for
Integers.
   use Integer_IO;

begin
   Open   (File => Input,  Mode => In_File,  Name => "input.txt");  --
Associate logical and
   Create (File => Output, Mode => Out_File, Name => "output.txt"); --
physical disk files.
                                                -- May use "string" variables
to hold file names.

   lf := integerToCharacter(lf_integer);  -- Convert ASCII integer to
character lf for I/O.

   for J in 1..4 loop
     get(Input, char);       -- Read character from file Input.
     put(char); put("  ");   -- Print on CRT (screen).
     put(Output, char);      -- Print in file Output.
   end loop;
   new_line; new_line;
   put(Output, lf); put(Output, lf);

   put("Test empName I/O:  ");
   nameIn := Sam;
   put(nameIn); put("  ");
   new_line(2);
   put(Output, lf); put(Output, lf);

   for J in 1.. 3 loop
     get(Input, nameIn);
     put(nameIn); put("  ");
     put(Output, nameIn); put(Output,"  ");
```

```
   end loop;
   new_line(2);
   put(Output, lf); put(Output, lf);

   put(Output, "Boolean operation and I/O:  ");
   -- Boolean operation and I/O
   put("Boolean operation and I/O:  ");

   bool1 := True;
   bool2 := False;
   question := ((bool1 and bool2) or False) xor True;
   put(question);
   put(Output, question);

   Close (Input);
   Close (Output);

exception  -- Close files if there is a major error.
   when End_Error =>
      if Is_Open(Input) then
         Close (Input);
      end if;
      if Is_Open(Output) then
         Close (Output);
      end if;
end Using_Text_Files;
```

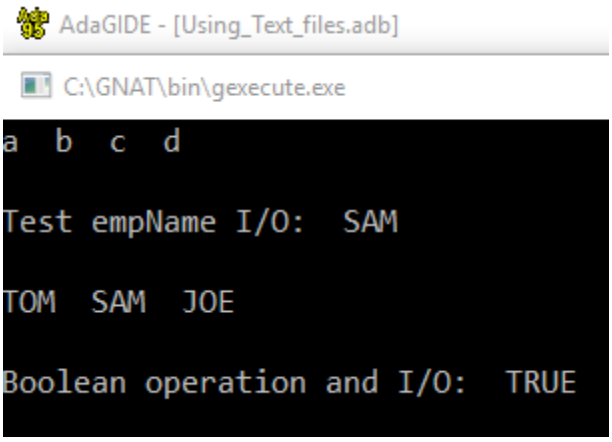
Contents of file input.txt:
abcd Tom
Sam Joe

Contents of file output.txt:
abcd

TOM SAM JOE

Boolean operation and I/O: TRUE

CRT/Screen:



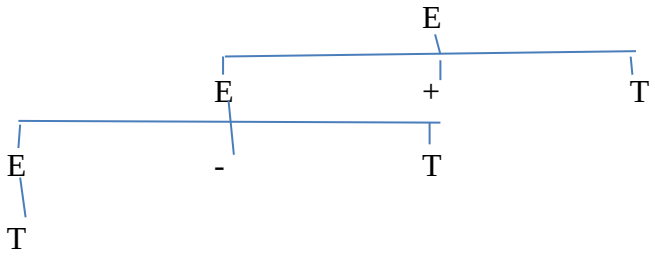
I recommend the most recent edition of "Programming in ADA" by John Barnes, for example ISBN 0-201-34293-6 for the second edition. All editions of Branes text starting with the 2nd edition are sufficient for the course. Almost any Ada book will work (Ada 2017, Ada 2014, Ada 2012, Ada 2010, or Ada 95). I recommend "Java How to Program" (6th edition or latter) by Deitel and Deitel for Java, ISBN 0-13-148398-6. "C/C++" text covering templates and inheritance will be adequate for the material covered in labs. Select a "C/C++" text oriented to the platform on which you plan to code, i.e., Windows, Linux, Chrome, etcetera.

Please remember use of work by other students from this semester or previous semesters will generally result in an "F" for the course! You are however encouraged to discuss solutions with each other and provided reasonable help. One of the best ways to learn is from your fellow students or helping fellow students. You may not however work together as a team to solve lab problems.

Application to Language Translators. Consider the following grammar which describes any expression with addition or subtraction forcing left to right evaluation.
E := E + T | E – T
E := T
Where a "T" is a variable or constant.

E := E + T
:= E - T + T
:= T - T + T

Derivation tree:



Warshall's Algorithm for finding R^+

[Published in 1962]

Assume an N by N Boolean Matrix Representation for a transitive Boolean relation. Warshall's Algorithm generates the transitive closure.

```
for I := 1, 2, °°° , N loop
      for J := 1, 2, °°° , N loop
         if A[J,I] = true then
              for K := 1, 2, °°° N loop
                   A[J,K] := A[J,k] or A[I,K];
              end Loop;
         end if;
      end loop;
end loop;
```

Short cut by hand:

1) or the i^{th} row and j^{th} row and store in row i.
2) Scan row i of M, if element j in row i = 1, then or row j into row i producing a new row i. Stop when an entire pass produces no new 1's for all I and j.

$$\begin{bmatrix} 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix} \Rightarrow \begin{bmatrix} 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

$$\begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 1 \end{bmatrix} \overset{\text{ROW 2 TO 1}}{\Rightarrow} \begin{bmatrix} 0 & 1 & 1 \\ 0 & 0 & 1 \\ 1 & 0 & 1 \end{bmatrix} \overset{\text{ROW 3 TO 1}}{\Rightarrow} \begin{bmatrix} 1 & 1 & 1 \\ 0 & 0 & 1 \\ 1 & 0 & 1 \end{bmatrix} \overset{\text{ROW 3 TO 2}}{\Rightarrow} \begin{bmatrix} 1 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 0 & 1 \end{bmatrix} \overset{\text{ROW 1 TO 3}}{\Rightarrow} \begin{bmatrix} 1 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 1 \end{bmatrix} \overset{\text{ROW 1 TO 2}}{\Rightarrow} \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$