CSCI 2011: Mathematical Relations

Chris Kauffman

Last Updated:
Thu Jul 26 09:50:10 CDT 2018

Logistics

Reading: Rosen

- Now: 9.1 - 9.5
- Now/Next: 10.1 - 10.3

Goals

- Finish up Recurrence Relations
- Discuss Mathematical Relations
- Intro Graph Theory

Assignments

- A07: Due Today
- A08: Post Thu
- **Last assignment**

Quiz 5 Thursday

- Discrete Probability
- Recurrence Relations
- **Last Quiz**

Mathematical Relations

- **Very** general idea: a **relation** defines items that are related (duh)
- Most of our attention will be on relations between pairs of objects, **binary** relations on a set
- Lends itself to several interesting representations/algorithms involving
  - Matrices and multiplication
  - Graphs and paths

N-ary Relations

- Use *N*-tuples to represent multiple related pieces of information
- Studied in Database Courses
- Usually discuss several relations or **tables** and associated operations such as **querying** and **joining**
- Database access typically uses special languages such as **Structured Query Language (SQL)**
- Best drawn as tables as other representations don't lend much insight

Name	ID	Major	GPA
(Ackermann,	231455,	Computer Science,	3.88)
(Adams,	888323,	Physics,	3.45)
(Chou,	102147,	Computer Science,	3.49)
(Goodfriend,	453876,	Mathematics,	3.45)
(Rao,	678543,	Mathematics,	3.90)
(Stevens,	786576,	Psychology,	2.99)

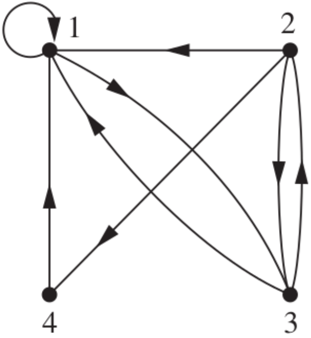
Binary Representations, Matrices, Graphs

- A **binary relationship on a Set S** is described as a set of pairs

  $$R = \{(a_1, b_1), (a_2, b_2), \ldots, (a_n, b_n)\}$$

  where each of a_i, b_i are in S
- **Matrix Repr. of Relations**
  - Rows for a_i, Cols for b_i
  - 0/1 Matrix: $x_{ij} = 1$ if $(a_i, b_j) \in R$
- **Graph Repr. of Relations**
  - Nodes for each a_i, b_j
  - (a_i, b_j) gives directed edge $a_i \to b_j$

	1	2	3	4
1	1	0	1	0
2	1	0	1	1
3	1	1	0	0
4	0	0	0	0



Example: Matrix/Graph for R

$$R = \{(1,1), (1,3), (2,1), (2,3), (2,4), (3,1), (3,2)\}$$

Exercise: Show the Matrix and Graph

- Ask 5 friends which friends they prefer to study with

$$R = \{(Al, Barb), (Al, Cole), (Al, Diane), (Barb, Cole), (Barb, Diane), (Cole, Cole), (Diane, Barb), (Diane, Cole), (Ellen, Barb)\}$$

- Draw the matrix and graph representations for this relation

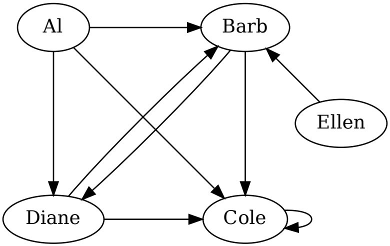
Answers: Show the Matrix and Graph

- Ask 5 friends which friends they prefer to study with

$$R = \{(Al, Barb), (Al, Cole), (Al, Diane), (Barb, Cole), (Barb, Diane), (Cole, Cole), (Diane, Barb), (Diane, Cole), (Ellen, Barb)\}$$

- Draw the matrix and graph representations for this relation

	A	B	C	D	E
A	-	1	1	1	-
B	-	-	1	1	-
C	-	-	1	-	-
D	-	1	1	-	-
E	-	1	-	-	-



Common Types of Relations

Relation R on a set A so comprised of pairs (a_i, a_j) with $a_i \in A, a_j \in A$. Several common properties of relations of note.

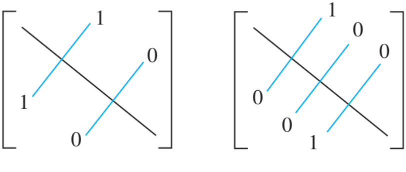
Reflexive Relations

- R is Reflexive if $(a_i, a_i) \in R$ for each $a_i \in A$
- Induces a Matrix representation with **main diagonal** all 1's
- All nodes in graph have self-loops

$$\begin{bmatrix} 1 & & & & & \\ & 1 & & & & \\ & & 1 & & & \\ & & & \ddots & & \\ & & & & 1 & \\ & & & & & 1 \end{bmatrix}$$

Symmetric Relations

- R is Symmetric $(a_i, a_j) \in R$ implies $(a_j, a_i) \in R$
- Induces a **symmetric matrix** representation: upper right triangle mirrors lower left
- All edges in graph are two-way



(a) Symmetric (b) Antisymmetric

Closures

- Suppose R is NOT Reflexive
- Add as a few pairs to it as possible to make it a Reflexive relation
- Called the **Reflexive Closure** of R
- Similarly, **Symmetric Closure** of R adds as few pairs as possible to make it a Symmetric relation
- Generally, the **closure** of a relationship R with respect to a property P is formed by adding as few pairs as possible so that R has property P
- Fairly easy for reflexivity and symmetry, but there's one other property that is more interesting

Transitive Relationships

R is a transitive relationship if it satisfies the following

- If $(a, b) \in R$ and $(b, c) \in R$
- Then $(a, c) \in R$

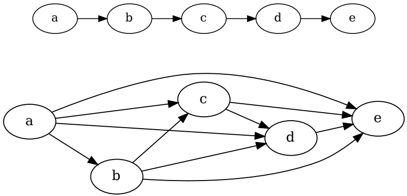
Leads to an interesting problem of forming the **Transitive Closure**

- If $R = \{(a, b), (b, c), (c, d), (d, e)\}$
- It's transitive closure is

$$R^* = \{(a, b), (a, c), (a, d), (a, e), (b, c), (b, d), (b, e), (c, d), (c, e), (d, e)\}$$

R	a	b	c	d	e
a	-	1	-	-	-
b	-	-	1	-	-
c	-	-	-	1	-
d	-	-	-	-	1
e	-	-	-	-	-

R^*	a	b	c	d	e
a	-	1	1	1	1
b	-	-	1	1	1
c	-	-	-	1	1
d	-	-	-	-	1
e	-	-	-	-	-



Transitive Closures

- In a directed graph, Transitive Closures reveals which nodes are **reachable** from starting points via a **path**
- Sometimes called the **connectivity** of the relation or the **reahability** graph
- Useful in transportation and routing
  - Can a series of flights from city A reach city X?
  - Can taking a series of buses from stop A reach destination D?
- Several algorithms exist to compute the transitive closure

Multiplication Algorithm for Transitive Closure

- Below is algorithm `trans_clos()` along with helper `boolean_matprod()`
- Exploits powers of matrix rep of *Rel*
  - $Rel = Rel^1$: reachable via 1 hop
  - $Rel^1 \times Rel = Rel^2$: reachable via 2 hops
  - $Rel^2 \times Rel = Rel^3$: reachable via 3 hops

```
bool[][] trans_closure(bool Rel[][])
{
  assert(Rel is a square matrix);
  int n = rows(Rel);
  bool[][] Pow = copy(Rel);
  bool[][] Clo = copy(Rel);
  for(int i=2; i<=n; i++){
    Pow = boolean_matprod(Pow,Rel);
    Clo = boolean_mator(Clo,Pow);
  }
  return Clo;
}
```

```
bool[][] boolean_matprod(bool X[][],
                         bool Y[][])
{
  assert(X,Y are square matrices);
  int n = rows(X);
  bool Z[][] = new matrix[n][n];
  set all elements of Z to false;
  for(int i=0; i<n; i++){
    for(int j=0; j<n; j++){
      for(int k=0; k<n; k++){
        bool b = X[i][k] AND Y[k][j];
        Z[i][j] = Z[i][j] OR b;
      }
    }
  }
  return Z;
}
```

Demonstration of `trans_clos()`

$$Rel = \{(a,c),(b,c),(c,e),(d,a),(d,e),(e,d)\}$$

- Make use of a nice matrix environment like Octave
- Allows matrix mult with * and matrix or with |

```
GNU Octave, version 4.4.0
> Rel=[[0 0 1 0 0];
       [0 0 1 0 0];
       [0 0 0 0 1];
       [1 0 0 0 1];
       [0 0 0 1 0]];
> Rel=logical(Rel);
> Pow=Rel;
> Clo=Rel;
```

```
> i=2;
> Pow = logical(Pow*Rel)  > Clo = Clo | Pow
  0  0  0  0  1             0  0  1  0  1
  0  0  0  0  1             0  0  1  0  1
  0  0  0  1  0             0  0  0  1  1
  0  0  1  1  0             1  0  1  1  1
  1  0  0  0  1             1  0  0  1  1
> i=3;
> Pow = logical(Pow*Rel)  > Clo = Clo | Pow
  0  0  0  1  0             0  0  1  1  1
  0  0  0  1  0             0  0  1  1  1
  1  0  0  0  1             1  0  0  1  1
  1  0  0  0  1             1  0  1  1  1
  0  0  1  1  0             1  0  1  1  1
> i=4;
> Pow = logical(Pow*Rel)  > Clo = Clo | Pow
  1  0  0  0  1             1  0  1  1  1
  1  0  0  0  1             1  0  1  1  1
  0  0  1  1  0             1  0  1  1  1
  0  0  1  1  0             1  0  1  1  1
  1  0  0  0  1             1  0  1  1  1
> i=5;
> Pow = logical(Pow*Rel)  > Clo = Clo | Pow
  0  0  1  1  0             1  0  1  1  1
  0  0  1  1  0             1  0  1  1  1
  1  0  0  0  1             1  0  1  1  1
  1  0  0  0  1             1  0  1  1  1
  0  0  1  1  0             1  0  1  1  1
```

Exercise: Transitive Closure Algorithm

- Construct pseudocode for the `boolean_mator(X,Y)` function
- Analyze the runtime complexity of `boolean_mator(X,Y)`
- Analyze the runtime complexity of `boolean_matprod(X,Y)`
- Analyze the runtime complexity of `trans_closure(Rel)`
- Analyze the **space** complexity of `trans_closure(Rel)`

```
bool[][] trans_closure(bool Rel[][])
{
  assert(Rel is a square matrix);
  int n = rows(Rel);
  bool[][] Pow = copy(Rel);
  bool[][] Clo = copy(Rel);
  for(int i=2; i<=n; i++){
    Pow = boolean_matprod(Pow,Rel);
    Clo = boolean_mator(Clo,Pow);
  }
  return Clo;
}
```

```
bool[][] boolean_matprod(bool X[][],
                         bool Y[][])
{
  assert(X,Y are square matrices);
  int n = rows(X);
  bool Z[][] = new matrix[n][n];
  set all elements of Z to false;
  for(int i=0; i<n; i++){
    for(int j=0; j<n; j++){
      for(int k=0; k<n; k++){
        bool b = X[i][k] AND Y[k][j];
        Z[i][j] = Z[i][j] OR b;
      }
    }
  }
  return Z;
}
```

Answers: Transitive Closure Algorithm

`n`: rows/cols of matrices

```
bool[][] bool_mator(bool X[][]
                    bool Y[][])
{
  assert(X,Y are square matrices);
  int n = rows(X);
  bool[][] Z = new matrix[n][n];
  for(int i=0; i<n; i++){
    for(int j=0; j<n; i++){
      Z[i][j] = X[i][j] OR Y[i][j];
    }
  }
  return Z;
}
```

- Doubly nested loop in `bool_mator(X,Y)` gives runtime $O(n^2)$

- `bool_matprod(X,Y)`: Triply nested loop gives runtime $O(n^3)$
- `trans_closure(Rel)` nests `bool_matprod()` in a loop of `n` iterations so it is $O(n^4)$
- `trans_clos(Rel)`: Space complexity is at least $O(n^3)$ for `Pow, Clo` matrix copies
- Avoiding repeated copies of matrices will improve performance

Exercise: Warshall's Algorithm

- Introduces notion of **interior nodes** on a path
  - To get from *a* to *c*, go through *b*
  - *b* is an interior vertex
- Repeatedly updates a matrix by determining whether a node appears on the interior of a path
- Runtime complexity of Warshall's Algorithm?
- Space complexity of Warshall's Algorithm?

```
bool[][] warshall_tc(bool Rel[][])
{
  assert(Rel is a square matrix);
  int n = rows(Rel);
  bool[][] Clo = copy(Rel);
  for(int v=0; v<n; v++){
    for(int i=0; i<n; i++){
      for(int j=0; j<n; j++){
        bool b = Clo[i][v] AND Clo[v][j];
        Clo[i][j] = Clo[i][j] OR b;
      }
    }
  }
  return Clo;
}
```

Exercise: Warshall's Algorithm

- Runtime complexity of Warshall's Algorithm?
  - Triply nested loop: $O(n^3)$
- Space complexity of Warshall's Algorithm?
  - $O(n^3)$ for copy `Clo` but works on this in place subsequently

```
bool[][] warshall_tc(bool Rel[][])
{
  assert(Rel is a square matrix);
  int n = rows(Rel);
  bool[][] Clo = copy(Rel);
  for(int v=0; v<n; v++){
    for(int i=0; i<n; i++){
      for(int j=0; j<n; j++){
        bool b = Clo[i][v] AND Clo[v][j];
        Clo[i][j] = Clo[i][j] OR b;
      }
    }
  }
  return Clo;
}
```

Demo of Warshall's Algorithm

$$Rel = \{(a,c),(b,c),(c,e),\\ (d,a),(d,e),(e,d)\}$$

- `v=-1`: Path between nodes with no interior vertices
- `v=0`: Path between nodes with $a = v_0$ as an interior AND all prior paths
- `v=1`: Path between nodes with $b = v_1$ as an interior AND all prior paths
- `v=2`: Path between nodes with $c = v_2$ as an interior AND all prior paths

```
v = -1
    0  1  2  3  4
 +--------------
0|  0  0  1  0  0
1|  0  0  1  0  0
2|  0  0  0  0  1
3|  1  0  0  0  1
4|  0  0  0  1  0

v =  0
    0  1  2  3  4
 +--------------
0|  0  0  1  0  0
1|  0  0  1  0  0
2|  0  0  0  0  1
3|  1  0  1  0  1
4|  0  0  0  1  0

v =  1
    0  1  2  3  4
 +--------------
0| 0  0  1  0  0
1| 0  0  1  0  0
2| 0  0  0  0  1
3| 1  0  1  0  1
4| 0  0  0  1  0
```

```
v =  2
    0  1  2  3  4
 +--------------
0| 0  0  1  0  1
1| 0  0  1  0  1
2| 0  0  0  0  1
3| 1  0  1  0  1
4| 0  0  0  1  0

v =  3
    0  1  2  3  4
 +--------------
0| 0  0  1  0  1
1| 0  0  1  0  1
2| 0  0  0  0  1
3| 1  0  1  0  1
4| 1  0  1  1  1

v =  4
   0  1  2  3  4
 +--------------
0| 1  0  1  1  1
1| 1  0  1  1  1
2| 1  0  1  1  1
3| 1  0  1  1  1
4| 1  0  1  1  1
```

Equivalence Relations

- **Equivalence Relations** have all 3 major relation properties
  - Reflexive, Symmetric, Transitive
- Most notable of Equivalence relation is **Congruence Modulus m** defined as

$$R = \{(a, b) | (a \textbf{ mod } m) = (b \textbf{ mod } m)\}$$

or (a, b) are related if they have equal values $\textbf{mod } m$

Example: The Caesar Cipher

- Encrypted messages by shifting forward modulo the size of the alphabet.
- With 26 letters, keys (0,26) are equivalent, (1,27) equivalent, etc
- $\{1, 27, 53, 79, \ldots\}$ are one **Equivalence Class** of the relation
- $\{2, 28, 54, 80, \ldots\}$ are another **Equivalence Class** of the relation
- 26 total equivalence classes for Caesar + 26 Letter Alphabet

Exercise: Code Induces Equivalence Classes

- Code to the right takes 3 ints
- Different inputs induce different **execution paths** through the code
  - **Path 1:** Line 4 executes, Line 7 doesn't
  - **Path 2:** ??
  - ...
- How many different paths are there?
- Give inputs that trigger each path

```
1   int max3(int a, int b,
2            int c)
3   {
4     int m = a;
5     if(b > m){
6       m = b;
7     }
8     if(c > m){
9       m = c;
10    }
11    return m;
12  }
```

Answers: Code Induces Equivalence Classes

Path	Line 4	Line 7	Input	Input	
1	Yes	Yes	1,2,3	1,2,4	...
2	Yes	No	1,3,2	1,4,2	...
3	No	Yes	2,1,3	2,1,4	...
4	No	No	3,2,1	4,2,1	...

- Software testing often tries for **coverage**:
  - Case coverage: inputs that hit all conditionals
  - Path coverage: inputs that hit all paths
- Code **partitions** inputs into equivalence classes on paths

```
1  int max3(int a, int b,
2           int c)
3  {
4    int m = a;
5    if(b > m){
6      m = b;
7    }
8    if(c > m){
9      m = c;
10   }
11   return m;
12 }
```