Maggie Johnson Handout #19
CS109

Relations

Key topics:

- * Introduction and Definitions
- * Graphs and Relations
- * Properties of Relations
- * Equivalence Relations
- * Partial Orderings
- * Composition of Relations
- * Matrix Representation
- * Closures
- * Topological Sorting
- * PERT and CPM

Suppose we have a set of Greek deities:

G = { Zeus, Apollo, Cronus, Poseidon }

As everyone knows, Zeus is the father of Apollo, Cronus is the father of Poseidon, and Cronus is also the father of Zeus. It seems that there exists some combination of the elements of G that satisfy the "is the father of" relation. To express this more precisely:

> A **relation** R between two sets A and B is a subset of A X B . In other words, A X B produces a set of ordered pairs <*a*, *b*>, *a* coming from set A, and *b* from set B. Some of these pairs will be "interesting", i.e., will satisfy our relation. For these pairs, we can write *a R b*, where R is the symbol for our relation. Alternatively, we can write <*a*, *b*> R . (Note that R is a generic relation symbol. Some relations, such as "less-than" have their own symbol: <.)

To be precise, what we have defined is a **binary relation**, so called because it operates on ordered pairs. We can also define **unary relations,** which operate on single elements, or ternary relations, which operate on ordered triples. In general an **n-ary relation** will operate on n-tuples.

Example 1

Let's consider the “is the father of” relation (which we will denote by F) on the set G X G. We can figure out that

G X G = { <Zeus, Zeus>, <Zeus, Apollo>, <Zeus, Cronus>,
<Zeus, Poseidon>, <Apollo, Zeus>, <Apollo, Apollo>,
<Apollo, Cronus>, <Apollo, Poseidon>, <Cronus, Zeus>,
<Cronus, Apollo>, <Cronus, Cronus>, <Cronus, Poseidon>,
<Poseidon, Zeus>, <Poseidon, Apollo>,
<Poseidon, Cronus>, <Poseidon, Poseidon> }

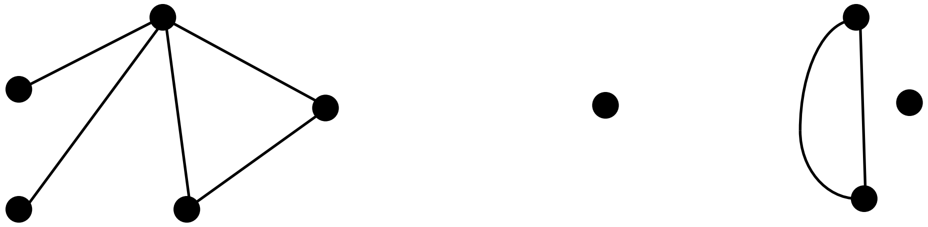
But of the set G X G, only a subset satisfies the "is the father of" relation. Thus, applying the F relation to G X G . yields the set:

{ <Zeus, Apollo>, <Cronus, Poseidon>, <Cronus, Zeus> }

meaning that Zeus *F* Apollo, Cronus *F* Poseidon, and Cronus *F* Zeus.

Graphs and Relations

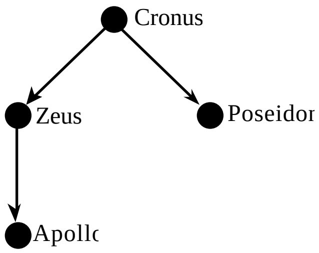
Graphs are a general representation for expressing many-to-many relationships. The easiest way to understand the definition of a graph is to look at a picture. The following are all examples of graphs:



Intuitively, we get the idea that a graph is a bunch of points connected by lines. The formal definition conveys this concept a bit more obtusely:

A **graph** is an ordered triple <*N*, *A*, *f* > where
N is a nonempty set of **nodes** or **vertices** (dots)
A is a set of **arcs** or **edges** (lines)
f is a function associating each arc *a* with an *un*ordered pair *x*, *y* of nodes
called the **endpoints** of *a*.

Graphs are incredibly useful structures, and the first use we will put them to is to represent the family relation described by the "father of" relation.



The astute reader may note that this graph has arrows rather than lines connecting the nodes. That is because the graph above is actually a special type of graph called a directed graph.

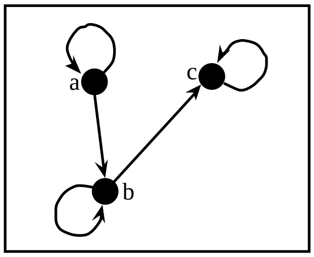
A **directed graph** (**digraph**) is an ordered triple <*N*, *A*, *f* > where
N is a nonempty set of nodes
A is a set of arcs
f is a function associating each arc *a* with an ordered pair <*x*, y> of nodes called the **endpoints** of *a*.

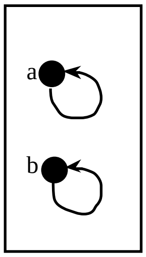
Directed graphs are very useful for representing binary relations, where the relation *a R b* is represented by drawing an arrow from *a* to *b*.

Properties of Relations

A relation is called **reflexive** on a set *S* if it satisfies *x R x* for any *x* in *S* That is,
<*x*, *x*> *R* for any *x* *S*.

Equality is reflexive on integers. For any integer *n* , it is always true that *n* = *n*. In pictures, these relations are reflexive:





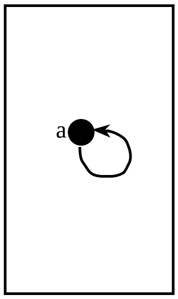
In a graph of a reflexive relation, every node will have an arc back to itself.

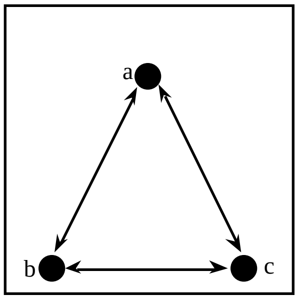
A relation is called **irreflexive** on a set *S* if *x R x* is not satisfied for every *x* in *S* That is, <*x*, *x*>   *R* for all *x*   *S*.

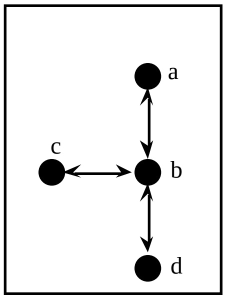
The less than relation is irrelexive on the integers. No number is less than itself.

A relation is **symmetric** on a set *S* if whenever *x R y* holds, *y R x* holds as well. That is, if *x*   *S* *and y*   *S and* <*x*, *y*>   *R* then <*y*, *x*>   *R*.

The "is a sibling of" relation is symmetric. If Apollo is a sibling of Artemis, then Artemis is a sibling of Apollo. It is easy to tell if a relation is symmetric by looking at it’s graph: it’s symmetric if every line has an arrowhead at either end.







A relation is **antisymmetric** on a set *S* if *x*   *S*, *y*   *S*, if <*x*, *y*>   *R*, and x ≠ y, then <*y*, *x*>   *R*,

The "is the father of" relation is antisymmetric. If Zeus is the father of Apollo, then certainly Apollo is not the father of Zeus (this doesn’t happen even in Greek mythology). In terms of a directed graph, whenever there is an arrow going out from an element, a relation is antisymmetric if there is not an arrow coming back to that element from the destination of the first arrow.

A relation is **transitive** on a set *S* if whenever
x, *y*, and *z* are in S, and
x R y holds, and
y R z holds, then
x R z holds as well.

That is, if *x*   *S*, *y*   *S*, *z*   *S*, <*x*, *y*>   *R* and <*y*, *z*>   *R* then <*x*, *z*>   *R*.

The "less-than" relation (<) is transitive. If $x < y$, and $y < z$, then it must be true that $x < z$.

Example 2...

Consider the following relations on {1, 2, 3, 4}:

r_1 : {<1,1>, <1,2>, <2,1>, <2,2>, <3,4>, <4,1>, <4,4> }
r_2 : {<1,1>, <1,2>, <2,1> }
r_3 : {<1,1>, <1,2>, <1,4>, <2,1>, <2,2>, <3,3>, <4,1>, <4,4> }
r_4 : {<2,1>, <3,1>, <3,2>, <4,1>, <4,2>, <4,3> }
r_5 : {<1,1>, <1,2>, <1,3>, <1,4>, <2,2>, <2,3>, <2,4>, <3,3>, <3,4>, <4,4> }
r_6 : {<3,4> }

Which relations are reflexive? ______________________________________
Which relations are symmetric? ______________________________________
Which are antisymmetric? ____
Which are transitive? ____

Consider the following proof: If a binary relation R is symmetric and transitive, it is also reflexive.

PROOF: Let x and y be members of the domain of R (which is symmetric and transitive). For any two elements x and y in R, if xRy we know that yRx, by symmetry. By transitivity, we know that xRy and yRx imply xRx. Since x is an arbitrarily chosen member of R's domain, we have shown that xRx for every element in the domain of R, and thus R is reflexive.

What do you think of this proof? ____

Equivalence Relations

These properties of relations can come in package deals, and these packages are given special names. A relation that is reflexive, symmetric,& transitive on a set *S* is called an **equivalence relation** on *S*.

Example 3

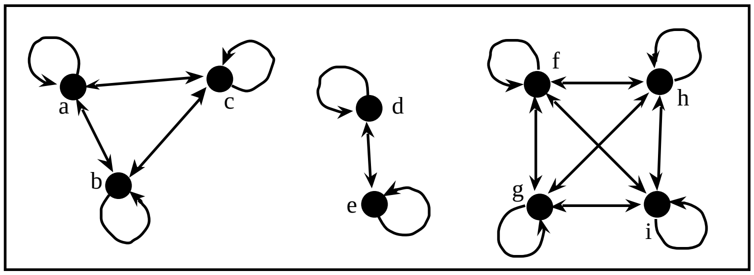
Suppose R is the relation on a set of strings such that a R b if and only if length(a) = length(b), where length(x) means length of string x.

Since length(a) = length(a), we can say a R a and therefore, R is reflexive. Suppose a R b, so length(a) = length(b). Then, length(b) = length(a) so R is also symmetric. Finally, suppose a R b and b R c; this means length(a) = length(b) and length(b) = length(c). We see that length(a) = length(c) so R is also transitive and is an equivalence relation.

The following relations are defined on the set of all people. Which are equivalence relations?

a) {(a,b) | a and b have met} ____________
b) {(a,b) | a and b were born in the same year} ____________
c) {(a,b) | a and b share a common parent} ____________

Graphs of equivalence relations are divided into “islands” of interconnected nodes, each separate subgraph representing an equivalence relation. These islands are called **partitions.** More formally, a partition of a set S is a collection of disjoint nonempty subsets of S that have S as their union.

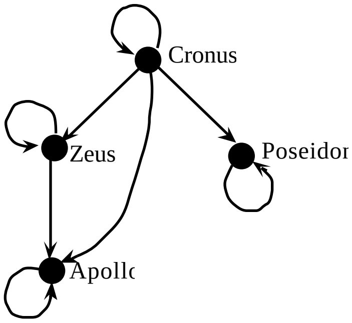


```
Consider the following proof:  Let R be a binary relation on a set A and suppose
R is symmetric and transitive; If, for every x in A, there is a y in A such that
xRy, then R is an equivalence relation.

PROOF: Suppose x is a particular but arbitrarily chosen element of A.  We know
there is a y such that xRy.  By symmetry, we know yRx and by transitivity, xRx.
Therefore, this relation is reflexive and as given, is also symmetric and
transitive.  Therefore, R is an equivalence relation.

What do you think of this proof? ____________________________________________
```

Partial Orderings

A relation *R* that is reflexive, antisymmetric, and transitive on a set *S* is said to define a **partial ordering** on *S*. A set *S* together with a partial ordering *R* is called a **paritally ordered set** or **poset**.

Consider the relation "is an ancestor of" on a set of Greek deities. We will define the relation in such a way that each person has himself as an ancestor as well as his parents, parents of parents, etc. Given this definition, it is clear that the "is an ancestor of" relation is reflexive and transitive. It is also clear that the "is an ancestor of" relation is antisymmetric. Therefore, this relation defines a partial ordering on the set of Greek deities.



Example 4

Show that ">=" is a partial ordering on the set of integers.

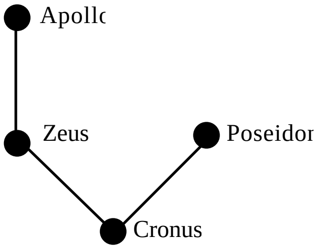
Since a >= a, this relation is reflexive. If a >= b and b >= a, then a = b which shows this relation is antisymmetric. If a >= b and b >= c, then a >= c so this relation is transitive. Thus, ">=" is a partial ordering on the set of integers.

Define a relation R on the set of integers as follows: For all integers m and n, mRn <=> every prime factor of m is a prime factor of n. Is this a partial order?

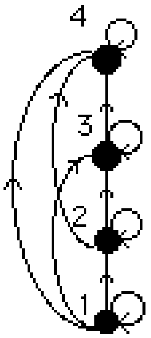
The graphs of partial orderings can be fairly complex. (Being both reflexive and transitive produces a lot of arcs.) One way to simplify these graphs is to use a special kind of representation known as a Hasse diagram.

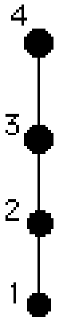
The general procedure for creating a Hasse diagram is as follows: 1) draw the directed graph for the relation; 2) remove all the loops at each node which must be there for reflexivity; 3) Remove all edges that must be there for transitivity; 4) Arrange each edge so that its initial node is below its terminal node as indicated by the directed edges; 5) Remove all arrows on the directed edges (since all edges point upward toward their terminal node).

The Hasse diagram for the ancestor relation is shown below:



Another example of a Hasse diagram for the partial ordering {<a,b> | a <= b } on {1,2,3,4} is given below along with its directed graph.





Because posets are at least semi-ordered, it is possible to pick out some extreme elements. In Hasse diagrams, these elements are easy to spot - they are the top and bottom elements:

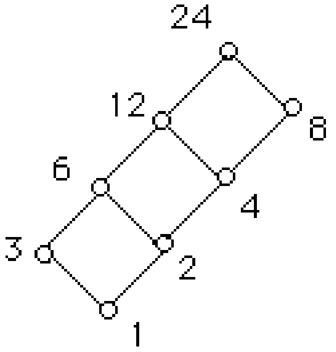
An element *a* of a poset (*S*, *R*) is **maximal** if it is not less than any element of the poset. In the “ancestors” example above, Apollo and Poseidon are maximal elements and Apollo is the **greatest element.**

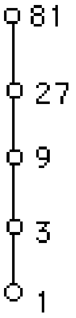
An element *a* of a poset (*S*, *R*) is **minimal** if it is not greater than any element of the poset. Cronus is the only minimal element in the “ancestors” example; it is also the **least element.**

A **total ordering** is a special case of a partial ordering where every element of the set is related to every other element. The Hasse diagram for a total ordering is about as simple as possible: it's a straight line. The example given above for <= is a total ordering. Suppose R is a partial ordering on a set A. Elements a and b of A are said to be **comparable** if and only if, either a R b or b R a. Otherwise a and b are **noncomparable**. R is a total ordering if all the elements in the relation are comparable.

Example 5

> The Hasse diagram for the partial ordering { <a,b> | a mod b = 0 } on the set of { positive divisors of 24 } is given below. This is not a total ordering because 6 and 4 (for example) are not comparable. The second diagram for the partial ordering { <a,b> | a mod b = 0 } on the set of { positive divisors of 81 } is a total ordering.



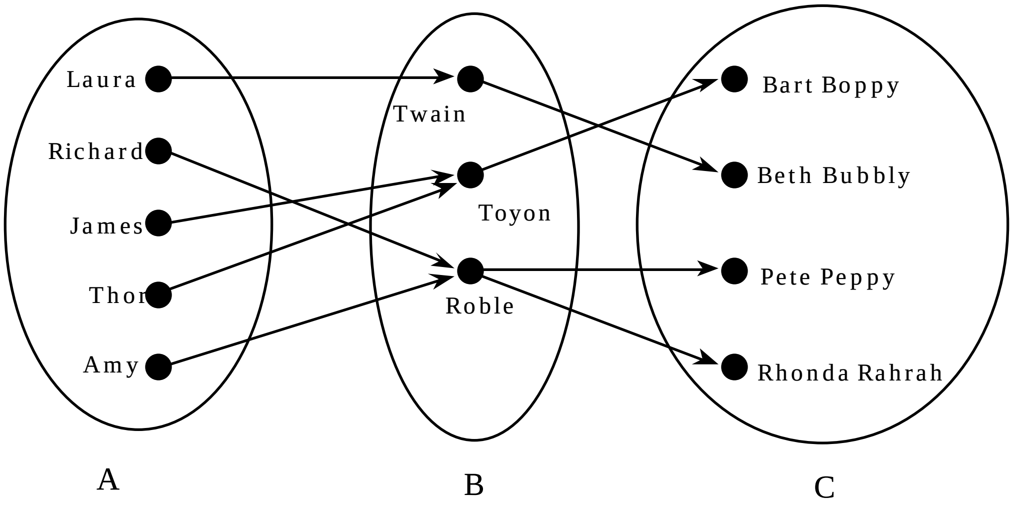


Consider the following relation R defined on the set S = {0, 1}: For all ordered pairs (a,b) and (c,d) in S x S, (a,b) R (c,d) | a <= c and b <= d. Is this a partial ordering? ________________ If so, what would the Hasse diagram look like?

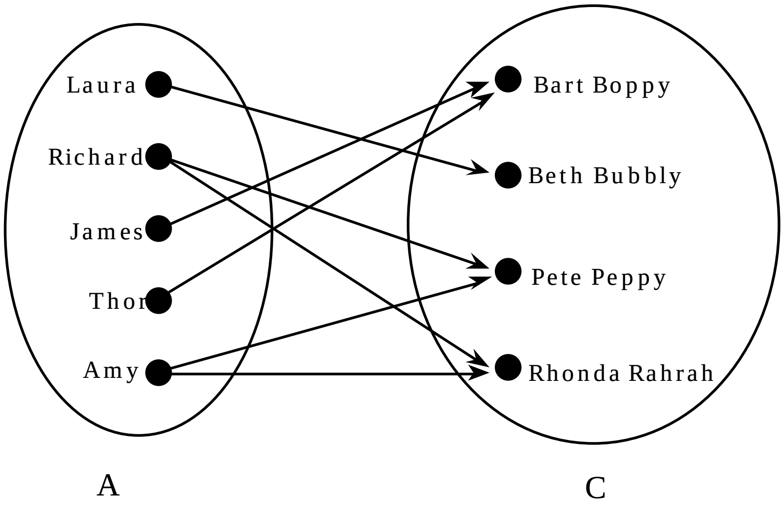
Composition of Relations

Suppose *R* is a relation on *A* X *B* and *S* is a relation on *B* X *C*. The **compostion** of *R* and *S*, denoted S o R, is a relation on *A* X *C* given by *S* o *R* = {<*a,c*> | *a* *A* , *c* *C*, and there is some *b* *B* such that *a R b* and *b S c*.}

This definition is rather dense, but the concept is very intuitive. The pictures below will make things more clear. In this example, we consider the relation *R* linking the set of students *A* with their dorms in *B* and the relation *S* linking dorms with Resident Assistants in *C*.



This represents the relation R on A X B and the relation S on B X C. The composition *S* o *R* links students in *A* with their RA's in *C:*



This represents the compostion S o R on A X C. The easy way to construct a composition S o R is for each pair of elements <*a,c*> (*a* A , *c* *C*), simply check if there is a path from *a* to *c* through any element of *B*. This is completely equivalent to the definition given above.

Example 6

Find S o R where R = {<1,1>, <1,4>, <2,3>, <3,1>, <3,4>} and S = {<1,0>, <2,0>, <3,1>, <3,2>, <4,1>}

Another way to look at composition of relations is: find the ordered pairs where the second element of the ordered pair in R matches the first element of the ordered pair in S. For example, <2,3> in R and <3,1> in S produce the ordered pair <2,1> in S o R.

S o R = { <1,0>, <1,1>, <2,1>, <2,2>, <3,0>, <3,1>}

Matrix Representation

For humans, it is often convenient to view a relation as a graph. Computers, on the other hand, are not very good at looking at pictures. The method of choice for a computer is to represent a relation as a matrix. A relation *R* from set *A* to set *B* will have |*A*| rows and |*B*| columns. A "1" in row *i* and column *j* of the matrix means that the relation holds for the i^{th} element of *A* and the j^{th} element of *B*; a "0" means the relation does not hold. (The order of elements in the sets *A* and *B* doesn't matter. We are just giving them an arbitrary but consistent numbering so that we know where to look in the matrix.)

Example 7

To represent the "father of relation" on the set of Greek dieties as a matrix, we will arbitrarily impose the following numbering on the set:

1 = Zeus
2 = Apollo
3 = Cronus
4 = Poseidon

Thus the matrix can be represented as :

$$\begin{bmatrix} 0\,1\,0\,0 \\ 0\,0\,0\,0 \\ 1\,0\,0\,1 \\ 0\,0\,0\,0 \end{bmatrix}$$

Example 8

A = {1,2,3}   B = {1,2}   R = relation from A to B containing <a,b> if a A, b B and a > b. What is the matrix representing R?

The ordered pairs of the relation R = {<2,1>, <3,1>, <3,2> } so the matrix is:

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

To find the composition of two relations, we can simply perform a variation on matrix multiplication. Rather than multiplying each individual pair of elements, we will take the logical "and", and rather than summing all the pairs, we will take the logical "or". This is easier to see with a diagram:

$$\begin{bmatrix} ?\,?\,? \\ a\,b\,c \\ ?\,?\,? \end{bmatrix} \begin{bmatrix} d\,?\,? \\ e\,?\,? \\ f\,?\,? \end{bmatrix} = \begin{bmatrix} ? & ? & ? \\ \begin{array}{l} \text{(a and d) or} \\ \text{(b and e) or} \\ \text{(c and f)} \end{array} & ? & ? \\ ? & ? & ? \end{bmatrix}$$

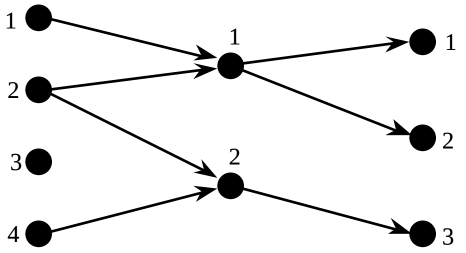
Computing row 2, column 1

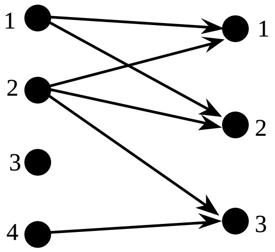
Each position in the answer matrix is computed using the same technique. Why does this technique work? There is no magic involved. The matrix product simply does mathematically what you do with your eyes when you look at the graph of a relation. It is just checking if there is a path from *a* to *c* (*a* *A* , *c* *C*) using any intermediate point in *B*. The "ands" check for a pair of connections <*a*,*b*> and <*b*,*c*>, and we are "or'ing" the results because any *b* *B* is OK.

Example 9

Let *R* = { <1,1>,<2,1>,<2,2>,<4,2> } and *S* = {<1,1>,<1,2>,<2,3> }. Find the composition *S* o *R*.

First we can draw the graph of the relations *R* and *S*. (This is not necessary, but it helps to show what is going on.) Next to the relations *R* and *S* we have drawn the composition *S* o *R*.





The matrix form of the relations is shown below.

$$\begin{bmatrix} 1\,0 \\ 1\,1 \\ 0\,0 \\ 0\,1 \end{bmatrix} \quad \begin{bmatrix} 1\,1\,0 \\ 0\,0\,1 \end{bmatrix} = \begin{bmatrix} 1\,1\,0 \\ 1\,1\,1 \\ 0\,0\,0 \\ 0\,0\,1 \end{bmatrix}$$

Closures

If a relation R on a set S fails to have a certain property, we may be able to extend R to a relation R* on S that does have that property. All this means is if we have a set of ordered pairs that represent a relation R, we can add ordered pairs to the relation so the relation becomes reflexive, symmetric, or transitive. If R* is the smallest such set of ordered pairs, R* is called the closure of R with respect to a particular property.

Example 10

Let S = {0, 1, 2, 3} and R = {<0,1>, <0,2>, <1,1>, <1,3>, <2,2>, <3,0> }; R is not reflexive, symmetric or transitive. To make this relation reflexive, we must add the ordered pairs <0,0> and <3,3>. These pairs along with the original pairs give us the closure of R with respect to reflexivity. Notice that this is the smallest set of ordered pairs for reflexivity to hold.

The closure of R with respect to symmetry is

{<0,1>, <0,2>, <1,1>, <1,3>, <2,2>, <3,0>, <1,0>, <2,0>, <3,1>, <0,3> }

Transitive closure is the same as finding the composition of R o R (but there is a little more to it than that as we shall see). We look for the ordered pairs where the second element of the ordered pair matches the first element of another ordered pair. Thus, the transitive closure of R:

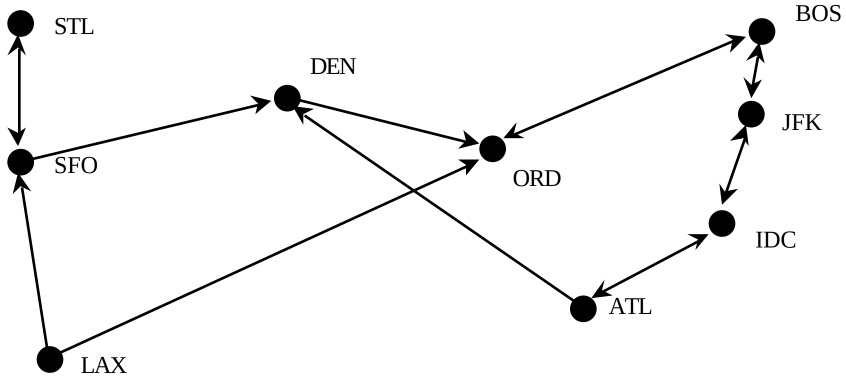
R U {<0,0>, <0,3>, <1,0>, <3,1>, <3,2>, <3,3>, <1,2>}

`If A is the set of all people and R means "is a child of", how would you describe the transitive closure of R?` ______________________________

`Can there be such a thing as antisymmetric closure?` ______________________________

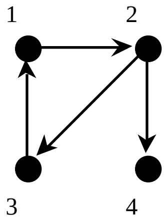
Transitive closures are especially useful in modeling certain types of problems. For example:

Erratic Airways has published the route map shown below. As an expert in such matters, you quickly realize that the map is actually a directed graph, and a directed graph is equivalent to a relation. The relation depicted below is “connected directly by Erratic Airways” on a set of cities.

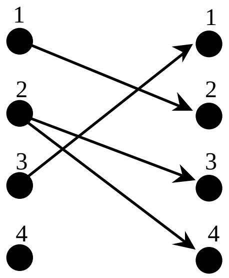


A more useful relation, however, might indicate whether it was possible to fly from one city to another by taking a sequence of flights rather than just one. In graph terms, the relation “connected by a sequence of flights” is equivalent to determining which pairs of nodes in the graph are connected by a path. The relation “connected by a sequence of flights” is really the transitive closure of the original relation. This is a more general interpretation than described above where we, in essence, were only looking for paths of length 2. The transitive closure is really all paths of any length.

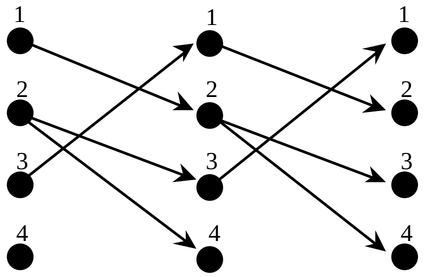
Computing the transitive closure of any reasonably sized relation by looking at the set of ordered pairs <*x*,*y*> can be a nightmare, and looking at the graph of the relation does not help much. For these tasks it is necessary to use some algorithm. Fortunately, we have one handy. (Note: the algorithm in this handout is not the best. We may see a more efficient algorithm called Warshall's algorithm, when we study graph theory.) Consider the relation *R* shown below on the set *S* = {1, 2, 3, 4}:



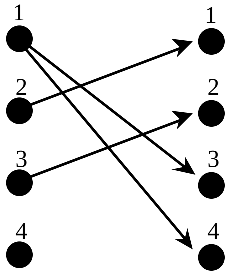
One way to make this transitive is to add arcs for all the paths of length 2, then all the paths of length 3, etc until there are no more paths to be added. But how can we we find the paths of length 2? If we redraw *R* to reflect the fact that it is a relation on *S* X *S* we see all the paths of length 1.



To find all the paths of length 2, we can redraw the relation R next to itself.



We can now find all the paths of length 2 by inspection of the graph above.



As noted earlier, finding the transitive closure is the same as finding the composition R o R. Thus R o R gives all paths of length 2, and is sometimes denoted as R^2. This means that R U R^2 contains all paths of length 2 or less. To find all paths of length 3, we compute $\mathrm{R}^3 = R$ o R o $R = R^2$ o $R = R$ U R^2 U R^3 Therefore, to find the transitive closure, we can use the following formula:

$$\mathrm{R}^* = R \text{ U } R^2 \text{ U } R^3 \ldots \text{ U } R^n \quad (\text{n } = \text{\# vertices})$$

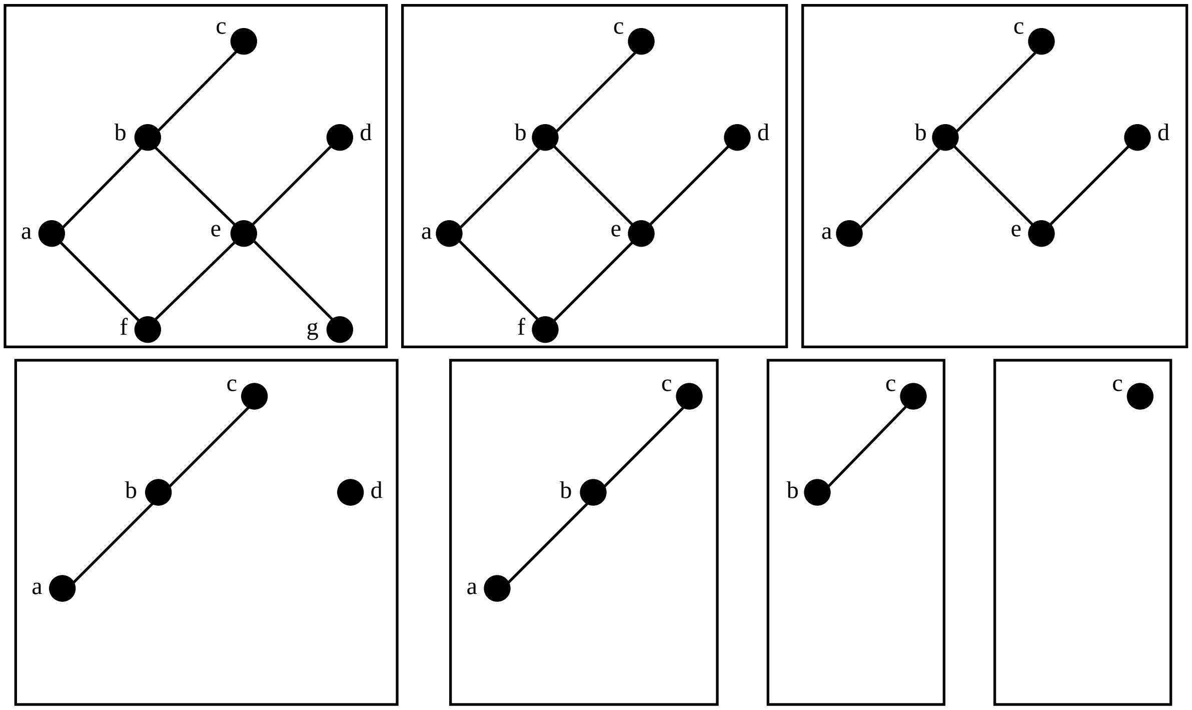
Still, this is probably more work than you want to do by hand, so it's nice to have the computer do it for you using matrices. We already know how to compose relations using matrices. To find the "square" of a matrix, you do matrix multiplication. Then, the union of two matrices is simply the result of "or'ing" the corresponding elements of each matrix. With this, it is possible to write a program that will find the transitive closure of any relation.

Let R be the relation on the set of all students containing the ordered pair (a,b) if a and b are in at least one common class and a <> b. When is (a,b) in R^2, R^3 and R^*? ____

Topological Sorting

Partial orderings have an immediate application in many scheduling algorithms. It is easy to think of projects where some jobs can be started only after some other jobs have been completed. (For example, the foundation of a building must be completed before the 30th story can be put on; before the foundation can be built, the site must be excavated, etc.) These projects can be thought of as posets, where elements of a set of jobs are partially ordered by the relation "must be completed before."

The Hasse diagram in the first box below shows such a poset. In this example, *f* must be completed before *a*, which must be completed before *b*, etc.



Extracting the total ordering g,f,e,d,a,b,c from a poset.

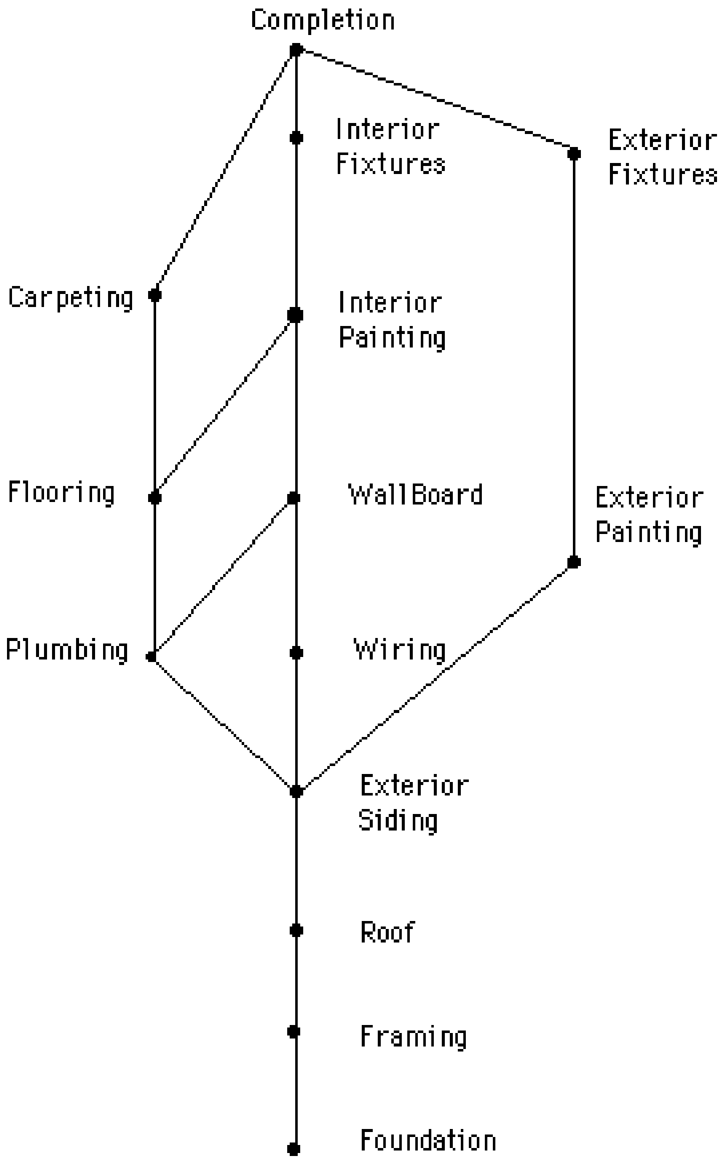
The trick to scheduling, of course, is to come up with a sequence of jobs that is consistent with the requirements of the project. In more formal terms, we are trying to find a total ordering that is compatible with a given partial ordering. The topological sort algorithm is a very straightforward solution. Given a poset, we find a minimal element. This is the first element in the total ordering. We then remove this element from the poset and repeat finding minimal elements until there is nothing left. The boxes above show the algorithm in action.

Think of this in terms of a real project, like baking a cake (or something like that):

g: put the bowl on the counter
f: get all the ingredients and put them on the counter
e: throw all the ingredients in the bowl
d: put the ingredients away
a: plug in the mixer
b: mix it all up
c: throw it in the oven

Notice that there may be more than one topological sort for any poset. We could have sorted: *g, f, a, e, d, b, c*. This would have worked just as well on the specified project.

Example



The above Hasse diagram represents the tasks needed to build a house. Do a topological sort to determine one possible order of tasks.

There are many compatible total orders. One possible answer: Foundation, Framing, Roof, Exterior Siding, Wiring, Plumbing, Flooring, Wallboard, Exterior Painting, Interior Painting, Carpeting, Interior Fixtures, Exterior Fixtures, Completion.

PERT and CPM

Two widely used applications of partial order relations are PERT (Program Evaluation and Review Technique) and CPM (Critical Path Method). These techniques came into being in the 1950s to deal with the complexities of scheduling the individual activities needed to complete very large projects. The techniques are similar, but they were developed independently. PERT was developed by the US Navy to help organize the construction of the Polaris submarine. CPM was developed by DuPont Chemicals for scheduling chemical plant maintenance. The following example illustrates how the techniques work.

Example 2

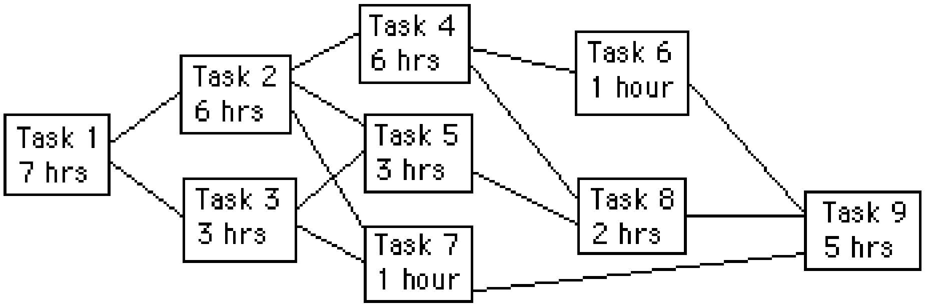
At an auto assembly plant, the job of putting together a car can be broken down into the following tasks:

1. build frame
2. install, engine, power train components, gas tank
3. install brakes, wheels, tires
4. install dashboard, floor, seats
5. install electrical lines
6. install gas lines
7. install brake lines
8. attach body to panels
9. paint body

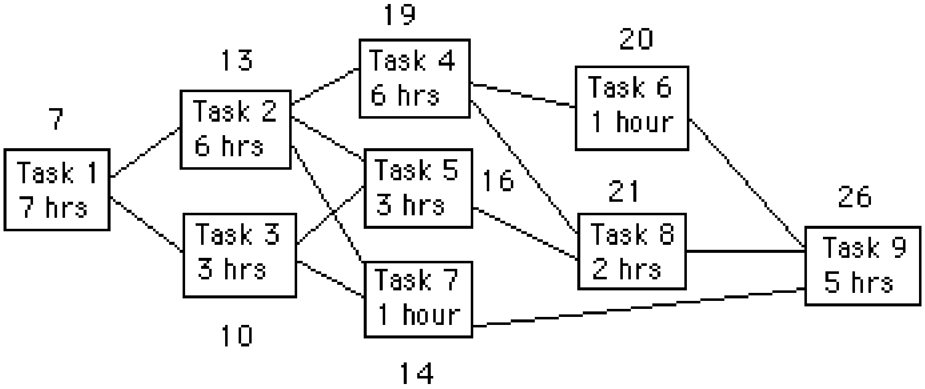
Some of these tasks can be done at the same time as others, while some cannot be started until others are finished. The following table summarizes:

Task	Immediately Preceding Tasks	Time to complete
1		7 hours
2	1	6 hours
3	1	3 hours
4	2	6 hours
5	2, 3	3 hours
6	4	1 hour
7	2, 3	1 hour
8	4, 5	2 hours
9	6, 7, 8	5 hours

We can build a Hasse diagram for this table, but the typical PERT or CPM representation is to turn the diagram sideways to reflect the chronological organization of tasks:



What is the minimum time required to build a car given unlimited resources to do the work? You can determine this by working from left to right across the diagram, noting for each task, the minimum time required to complete the task *starting from the beginning of the assembly process.*



Task 1 takes 7 hours; Task 2 requires the completion of task 1 plus 6 hours for itself. Similarly, task 3 takes 10 hours. Task 5 requires the completion of tasks 2 and 3 so the minimum time for task 5 = the time for task 5 itself + the maximum of the times to complete tasks 2 or 3 = 13 hours. So, task 5 takes 16 hours. Similarly, task 7 takes 14 hours, etc. 26 hours is the minimum time to complete the entire process which reflects the time to complete asks 1, 2, 4, 8, 9. This path is called the **critical path**.

Bibliography

Any good discrete math book has a section on relations. For more detailed information, refer to:

E.G. Coffman, *Computer and Job Scheduling Theory*, New York: Wiley, 1976.

M. Garey, R. Graham, D. Johnson, "Performance Guarantees for Scheduling Algorithms," *Operations Research*, Vol. 26, 1978.

R. Grimaldi, *Discrete and Combinatorial Mathematics*, *2nd ed.*, Reading, MA: Addison-Wesley, 1989.

K. Rosen, *Elementary Number Theory and its Applications, 2nd ed.*, Reading, MA: Addison-Wesley, 1988.

S. Sahni, *Concepts in Discrete Mathematics*, Minneapolis, MN: Camelot, 1985.

J.D. Wiest, F. Levy, *A Management Guide to PERT/CPM*, Englewood Cliffs, NJ: Prentice Hall, 1981.

Historical Notes

The concept of relations was first introduced by Karl Friedrich Gauss (1777-1855) in *Disquisitiones Arithmeticae* (which he wrote at the age of 24). In it, he discussed congruence modulo m which is defined as follows:

> If a and b are integers and m is a positive integer, then a is congruent to b modulo m if m divides a-b. This is notated $a \equiv b \pmod{m}$. In other words, $a \equiv b \pmod{m}$ if and only if a mod m = b mod m.

He also proved (although using somewhat different terminology) that congruence modulo m is an equivalence relation. As mentioned above, an equivalence relation splits the elements of a set into disjoint classes called partitions. There are m different congruence classes modulo m, corresponding to the m different remainders possible when an integer is divided by m. These classes form a partition of the set of integers. Consider for example, congruence modulo 4:

> remainder 0: {...., -8, -4, 0, 4, 8,....}
> remainder 1: {...., -7, -3, 1, 5, 9,....}
> remainder 2: {....,-6, -2, 2, 6, 10,...}
> remainder 3: {...., -5, -1, 3, 7, 11,..}

These classes are disjoint and every integer is in exactly one of them. This represents one of the first mappings of one set (the integers) to another (congruence modulo m) based on a relation.

Hasse Diagrams were named after Helmut Hasse (1898-1979), a German mathematician who introduced these diagrams in his 1926 textbook *Höhere Algebra*, as an aid to solving polynomial equations. Matrices were invented by Arthur Cayley (1821-1895) as a mechanism for representing "higher space" (space of n dimensions).