GB13604 - Maths for Computer Science

Lecture 4 – Graphs Part I

Claus Aranha
caranha@cs.tsukuba.ac.jp

College of Information Science

2024-10-30

Last updated October 28, 2024

Graphs – Lectures 4 and 5

Lecture I: Chapter 9

- Graphs and Relations
- Directed Graphs and Walks
- Scheduling and Partial Orders

Lecture II: Chapter 11

- Using Isomorphism
- Coloring and Connectivity
- Spanning Trees
- Matching

Part 1: Graph Definitions

Using Graphs to Solve Problems

First Example Problem: Course Registration

We can represent a list of courses in a university, and their requirements, using a graph structure. For example, to take *"Computer Graphics"*, you need to take *"Programming Theory"* and *"Linear Algebra"* first.

If you take two lectures per semester, how long would it take you to graduate? Why?

Code	Lecture	Prerequisites
0000	Social Questions	*none*
0001	Intro to Programming	*none*
0002	Calculus I	*none*
0003	Programming Theory	*0001*
0004	Linear Algebra	*0000, 0002*
0005	Programming Challenges	*0000, 0001, 0003*
0006	Computer Graphics	*0003, 0004*

Using Graphs to Solve Problems

First Example Problem: Course Registration

The university proposes a new lecture, *"Maths for Computer Science"*. The updated table is below.

How long does it take to graduate now? Why?

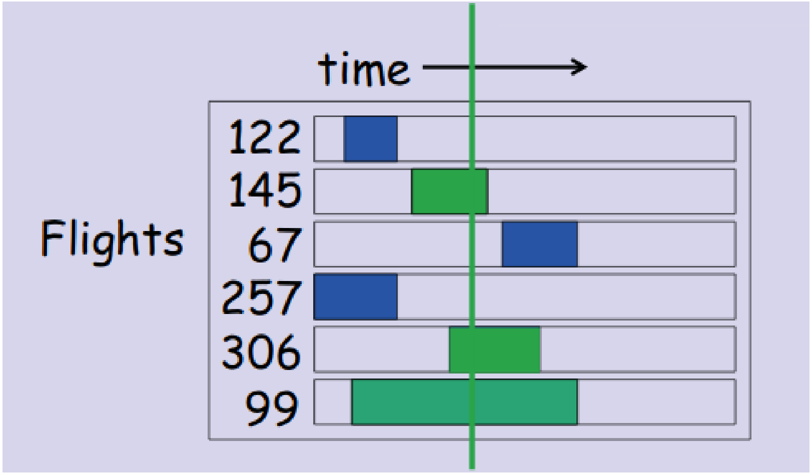
Code	Lecture	Prerequisites
0000	Social Questions	*none*
0001	Intro to Programming	*none*
0002	Calculus I	*none*
0003	Programming Theory	*0001*, **0007**
0004	Linear Algebra	*0000, 0002*
0005	Programming Challenges	*0000, 0001, 0003*
0006	Computer Graphics	*0003, 0004*
0007	**Maths for Computer Science**	***0005***

Using Graphs to Solve Problems

Second Example Problem: Airplane

In an airport, each airplane needs a gate when it is on the ground.

If we know the landing and departing time of each plane, what is the minimum number of gates necessary?



Problems as Graphs

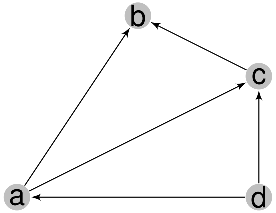
Many problems that can be described by the *relationship* between entities in the problem can be represented as graphs.

- A course is a prerequisite to another one;
- Two planes are landed at the same time;
- Webpages and the links between them;

These relationships can be described mathematically using a graph structure.

Graph representation allows us to use the same mathematical tools for different problems!

Graph Basic Definitions



A **graph** G is defined by a set of vertices V, and a set of edges E.

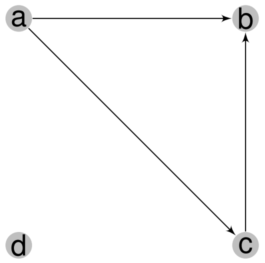
- $G = (V, E)$
- $V = \{a, b, c, d\}$
- $E = \{(a, b), (a, c), (c, b), (d, c), (d, a)\}$

A **directed graph (digraph)** is a graph where each edge has a **direction.**

An **undirected graph** is a graph where the edges do not have a direction ($(a, b) \iff (b, a)$).

Graphs and Relations

Remember Lecture 2?



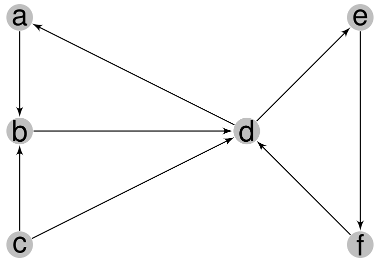
Graph $G = (V, E)$

- Set of vertices $V = \{a, b, c, d\}$
- Set of edges $E = \{(a, c), (a, b), (c, b)\}$

- Every graph can be described as a binary relation $V \to V$
    - $E(a) = \{b, c\}$
    - $E(c) = \{b\}$
    - $E(d) = \emptyset$

- And every binary relation can be described as a directed graph too!

Matrix Representation of a Digraph



	a	b	c	d	e	f
a	0	1	0	0	0	0
b	0	0	0	1	0	0
c	0	1	0	1	0	0
d	1	0	0	0	1	0
e	0	0	0	0	0	1
f	0	0	0	1	0	0

- A digraph is represented by a $|V|x|V|$ binary matrix (**adjacency matrix**)
- If there is an edge between v_i and v_j, then $A_{i,j} = 1$, else $A_{i,j} = 0$
- Another way to state it: $A(v_i, v_j) = 1 \iff v_j \in E(v_i)$

Part 2: Walks in Graphs

A solution to a problem can be a set of edges and vertices

Question: Is it possible to graduate in the curriculum below?

Code	Lecture	Prerequisites
0000	Social Questions	*none*
0001	Intro to Programming	*none*
0002	Calculus I	*none*
0003	Programming Theory	*0001*
0004	Linear Algebra	*0000, 0002*
0005	Programming Challenges	*0000, 0001, 0003*
0006	Computer Graphics	*0003, 0004*

Answer: Create the *curriculum's graph* and find a way to go to all vertices from the starting ones.

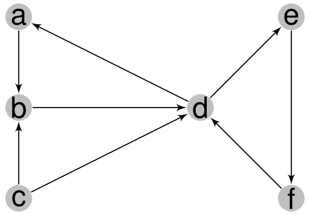
Walks and Paths

The solution of many graph problems is represented as a sequence of vertices and edges.

A sequence of vertices in a graph is a Walk or a Path.

- **Walk:** Any sequence of successive edges.
- **Path:** A walk that never visits the same vertex more than once.

Walk example

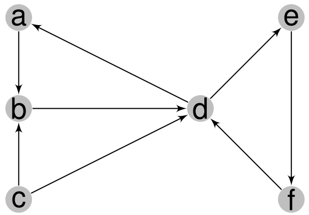


Walk: any sequence of successive edges

$$\begin{tikzcd} b \arrow[r] & d \arrow[r] & e \arrow[r] & f \arrow[r] & d \arrow[r] & e \end{tikzcd}$$

- **Walk lengh:** 5 edges (The length of a walk is the EDGES, not the VERTICES)
- **Representing as a compound relation:** $E(E(E(E(E(a)))))$
- NOTE: a walk can repeat edges and vertices!

Path example



Path: A walk without repeated vertices

$$\begin{tikzcd} e \arrow[r] & f \arrow[r] & d \arrow[r] & a \arrow[r] & b \end{tikzcd}$$

Stuck!

- **Path lengh:** 4 edges (The length is counted in EDGES, not vertices)
- **Eulerian Walk:** Every edge is visited once (**E**ulerian).
- **Hamiltonian Path:** Every vertex is visited once.
- **Eulerian / Hamiltonian Circuit**: You go back to the initial vertex

Proofs with Walks and Paths

Proof: The shortest Walk between two vertices is a Path.

Proof by contradiction: Assume a shortest walk that is not a path.

1. The shortest walk between v_0 and v_n is not a path. So it has a repeated vertice v_k:

$$w_{0,n} = v_0 \rightarrow v_1 \rightarrow \ldots \rightarrow v_k \rightarrow \ldots \rightarrow v_k \rightarrow \ldots \rightarrow v_{n-1} \rightarrow v_n$$

2. $w_{0,n}$ contains a smaller walk $w_{k,k}$ from v_k to v_k of size $|w_{k,k}| \geq 1$.
3. We can remove $w_{k,k}$ from $w_{o,n}$, resulting in a smaller walk $w'_{0,2}$ with size $|w_{0,n}| - |w_{k,k}|$
4. $w'_{0,n}$ is a walk from v_0 to v_n that is shorter than $w_{0,n}$, which is a contradiction.

$\square$

This can be used to prove the Triangle Inequality in walks (the distance of i, j is equal or smaller than the distance from $i, k + k, j$)

The Walk Relation

Let's describe a binary relation between vertices v_i and v_j in graph G, and call it the Walk Relation:

$$v_i G^n v_j.$$

The relation G^n means: "There is a walk from v_i to v_j with length exactly n."

Let's think about G^n:

- G^0 is the set of all vertices (walk of size 0 goes nowhere)
- G^1 is the set of all pairs of vertices connected by 1 edge
- **composition and addition:** $G^n \circ G^m = G^{n+m}$
  (For example, $G^2 + G^3 = G^5$)
- **common vertex:** If there is a composite relation between v_x and v_y ($v_x\ G^m \circ G^n\ v_y$), this implies that there is a common vertex connecting them:
  $v_x\ G^m \circ G^n\ v_y \rightarrow \exists z, v_x\ G^m\ v_k, v_k\ G^n\ v_y$

The Walk Relation and the Adjacency Matrix

Before, we described the Adjacency Matrix A, where $A_{i,j} = 1$ if $E(v_i, v_j)$.

If we rename A to A_{G^1}, we can generalize this concept to the Walk Matrix A_{G^n}, which is the adjacency matrix representing G^n: $A_{G^n,i,j} = 1$ if $v_i \; G^n \; v_j$. ($\exists$ walk of size n from v_i to v_j)

It is possible to calculate A_{G^n} using *boolean matrix multiplication*: $A_{G^n \circ G^m} = A'_{G^n} \odot A''_{G_m}$:

$$a_{ij} = A'_{i*} \odot A''_{*j} = (a'_{i0} \wedge a''_{0j}) \vee (a'_{i1} \wedge a''_{1j}) \vee (a'_{i2} \wedge a''_{2j}) \vee \ldots \vee (a'_{in} \wedge a''_{nj})$$

This means that we can calculate A_{G^n} quickly using the Fast Matrix Exponentiation:

- $A_{G^n} = A_{G^{n/2}} \odot A_{G^{n/2}} = (A_{G^{n/4}} \odot A_{G^{n/4}}) \odot (A_{G^{n/4}} \odot A_{G^{n/4}}) = \ldots$

Walk Relation G^* of a Digraph

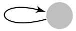
- G^n is the length n walk relation: It representes all walks of size exaclty n
- G^* is the walk relation of G: It representes all walks in G.
- uG^*v means that there is a walk of **some** length from u to v

QUIZ: How do we calculate A_{G^*} for a graph, given adjacency matrix A_{G^1}?

HINT: Remember that a walk can be infinite. But we need to solve this problem with a **finite** algorithm! When does the algorithm stop?

Walk Relation of a DiGraph

How to calculate G^*

1. Let G^1 be the relation defined by the original graph (or its adjacency matrix).
2. Let G^0 be the relation defined by the set V with self: 
   (A_{G^0}: the identity matrix).
3. The relation $G^{\leq 1} = G^0 \cup G^1$ is the relation for walks of *length 1 or less*;
   ($A_{G^{\leq 1}} = A_{G^0} \times A_{G^1}$)
4. $G^* = (G^{\leq})^{n-1}$: All walks of size $n-1$ or less. (**Q: Why not n?**)
5. Using fast exponentiation, you can calculate G^* in $\log n$ matrix boolean multiplications

Adjacency Matrices and Neural Networks

Why did we just study this?

We just studied an algorithm that can quickly calculate all the walks in a directed graph.

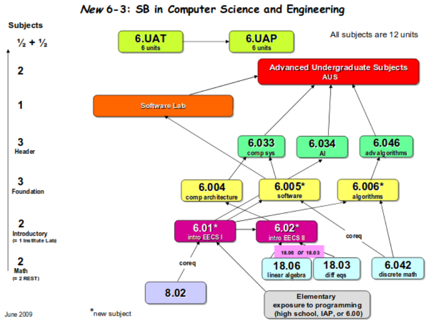
If the graph is also **acyclic**, and the edges have **weights** in them, this algorithm can be easily generalized to calculate the total weight between any two vertices.

This can be used to quickly calculate the weight between the input neuron and the output neuron of a very large network!

This is one reason why Matrix Multiplications are important for Machine Learning.

(caveat emptor: it gets more complicated with loops, non-linear activation functions, etc)

Walks and Prerequisite Course Graphs



The *walk relation* of the pre-requisite graph R defines how the courses relate to each other.

- Direct Prerequisite: Req(6.046) = 6.006
- Indirect Prerequisite: Req(6.046) = $\{6.006, 6.042, 6.01, 6.00, 8.02\}$

Course u is an indirect prerequisite of v if there is a positive length walk from u to v in R:

$$uD^{+}v$$

Requisites, Cycles and DAGs

In the beginning of the lecture, we talked about a prerequisite graph where it was not possible to graduate. Why? Because it had **cycles**.

- A closed walk is a walk that starts and ends at the same vertex.
- A cycle is a closed walk where the only repeat vertex is at the beginning and end.
  - $v_0 \rightarrow v_1 \rightarrow \ldots \rightarrow v_n \rightarrow v_0 | i > 0, j > 0, v_i \neq v_j$
  - A cycle is the path $v \rightarrow w + (w, v)$
- A Directed Acyclic Graph (DAG) is a digraph that has no positive length cycles.

Directed Acyclic Graphs Examples

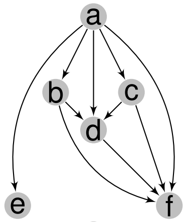
Directed Acyclic Graphs (DAGs), can be used to represent several ordered structures:

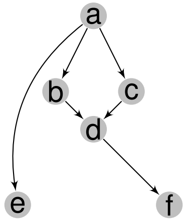
- Course Prerequisite Graphs;
- Ordered Task List:
  - "first add rice, then add water, then press cook button"
  - "Let x be 5, let y be 2, while y $>$ 0, multiply x by x and subtract 1 from y."

Computational structures can also be described using DAGs:

- Relations, for example:
  - Successor Relation: $n \to n+1$
  - Subset Relation: $\{1,2\} \subset \{1,2,3\}$
- Induction Proofs: $(P(n) \implies P(n+1) \implies P(n+2)\ldots)$;
- Dynamic Programming: (base cases and transitions on the DP table);

Directed Acyclic Graphs (DAG) and covering edges





Given a DAG *A*, its covering edges is the **smallest** DAG *B* that has the same Walk Relation as *A*

Walk relation of *A* and *B*:

- $a \rightarrow \{b, c, d, e, f\}$
- $\{b, c\} \rightarrow \{d, f\}$
- $d \rightarrow f$
- $\{e, f\} \rightarrow \emptyset$

Part 3: Scheduling and Partial Orders

Using DAGs for Scheduling

Let's consider again using DAGs for calculating course prerequisites.

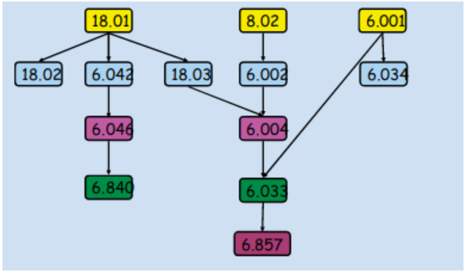
$18.01 \rightarrow 6.042$	$6.001 \rightarrow 6.034$	$6.001, 6.004 \rightarrow 6.033$
$18.01 \rightarrow 18.02$	$6.042 \rightarrow 6.046$	$6.033 \rightarrow 6.857$
$18.01 \rightarrow 18.03$	$8.02 \rightarrow 6.002$	$6.046 \rightarrow 6.840$
	$18.03, 6.002 \rightarrow 6.004$	

We say that *u* is a indirect prerequisite of *v* if there is a positive length walk in graph *R*:

$$18.01 \rightarrow 6.042 \rightarrow 6.046 \rightarrow 6.840$$

DAGs and Scheduling

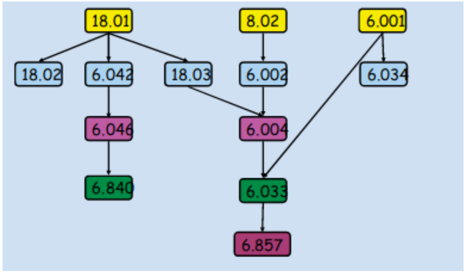
Minimal, Minimum, Maximal, Maximum of a DAG



- A minimal course is does not have any prerequisites:
  - $\emptyset \to$ 18.01, $\emptyset \to$ 6.001, $\emptyset \to$ 8.02
- A minimum course is an indirect prerequisite of **all** courses.
  - none in this example!
  - if we add a course $x \to \{18.01, 8.02, 6.001\}$, then x would be the minimum.
- Maximal and maximum courses have a similar definition.
  - $\{18.02, 6.840, 6.857, 6.034\} \to \emptyset$ are maximal.

DAG and Scheduling

How to Schedule



If we have the graph of course requirements, how do we select the courses for each semester?

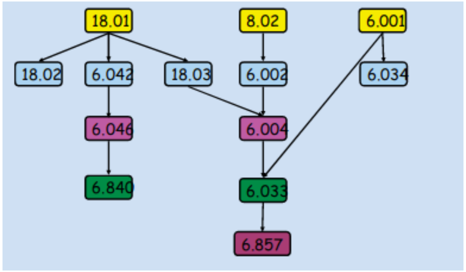
Greedy Scheduling:

1. Identify Minimal Subjects;
2. Add Minimal Subjects to Schedule;
3. Remove Minimal Subjects;
4. Return to Step 1

Schedule:
$\{18.01, 8.02, 6.001\} \rightarrow \{18.02, 6.042, 18.03, 6.002, 6.034\} \rightarrow \{6.046, 6.004\} \rightarrow \{6.840, 6.033\} \rightarrow 6.857$

DAG and Scheduling

Anti-Chains



- An anti-chain is a set of vertices (courses) where there is no direct or indirect requisite relation among them.
- This means that the courses in an anti-chain can be taken in any order, even all at the same time.
- Members of an anti-chain are imcomparable: It is not possible to say which one comes first.
- A relation graph can have multiple anti-chains. Example:
  - $\{6.046, 6.004\}$
  - $\{6.046, 18.03, 6.001\}$

DAG and Scheduling

Chains and Topological Sort

Chains

Just like anti-chain is a set of vertices that have no relation among themselves, a chain is a set of vertices that **all** have a relation among themselves.

Using of chains and anti-chains, we define a Topological Sort. A topological sort is an ordering of all vertices in G that obeys the requisite relations.

- 18.01, 6.001, 8.02, 6.002, 18.03, 6.034, 6.042, 18.02, 6.004, 6.046, 6.033, 6.840, 6.857
- 6.001, 8.02, 6.002, 18.01, 6.034, 18.03, 18.02, 6.042, 6.004, 6.046, 6.033, 6.857, 6.840

If G has anti-chains, it will also have multiple topological sorts.

DAG and Scheduling

Parallel Processing

We can use the same way of thinking to describe parallel scheduling of tasks.

- *n* tasks have to be executed by *p* processors.
- some pairs of tasks have a **prerequisite** relation.
- Minimum Parallel Time: minimum time to complete all tasks (assuming no limits on *p*)
    - Minimum Parallel Time = Maximum Chain Size
- Maximum Parallel Load: value of *p* necessary to achieve the Minimum Parallel Time
    - Maximum Parallel Load $\leq$ Maximum Anti-chain Size

Partial Orders and The Properties of a Relation

Until now, we observed scheduling (Partial Ordering) from the point of view of a DAG, describing the equivalent binary relation.

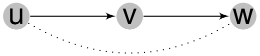
However remember that we can go the other way to: Every binary relation can be represented as a directed graph. And if that DiGraph is also acyclic, then we can define a Partial Order for the relation.

So, without explicitely building the graph, how can we tell if a binary relation has a partial order?

To answer that question, let's observe the properties of relations defined by DAGs.

Properties of Partial Orders 1: Transitivity

In a graph G, if there is a walk from u to v, and a walk from v to w, then there is a walk from u to w



This defines a transitive relation: $R(x,y)$ and $R(y,z)$ implies $R(x,y)$

Lemma: For any digraph G, the walk relations G^+ and G^* are transitive.

Definition: Transitive Relations

A relation **R** is transitive if: $xRy \wedge yRz \implies xRz$

Properties of Partial Orders 2: Irreflexibility

A binary relation R from set A to A is reflexive if $a\ R\ a$ for all $a \in A$.

In the same way, a binary relation R from set A to A is irreflexive if NOT($a\ R\ a$) for all $a \in A$.

We have defined the 0-length walk relation G^0 as the directed graph where every vertex is connected to itself. Therefore, the relations G^0 (and G^*) are reflexive.

On the other hand, we know that a directed graph is a DAG if it has **no cycle of positive lengths**. In other words, there is no path that starts in a vertice and goes back to itself.

Therefore, R is a DAG iff R^+ is irreflexive.

Strict Partial Orders

With these two properties, we can define when a Relation is a Strict Partial Order, and can be represented by A DAG:

A binary relation is a Strict Partial Order if it is **transitive** and **Irreflexive**

Examples of strict partial orders:

- The "Less than" relation: $a < b$
- The "Strict subset" relation: $A \subset B$

Path Total Orders

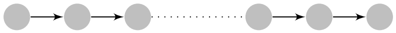
A Strict Partial Order is also Path Total if, for any two distinct elements, one will always be “greater than” another.

Example: The "Less than" relation $<$ on $\mathbb{R}$ is Path Total: for any two elements $x, y \in \mathbb{R}, x \neq y$, either $x < y$ or $y < x$

Counter-Example: The "Strict subset" relation is NOT Path Total. For example, subsets $\{2, 3\}$ and $\{1, 6\}$ are incomparable, regarding subsets.

- Relation R is path total: if $x \neq y \implies xRy \vee yRx$
- This means there are no imcomparable elements

In a path total relation, the graph representing it is a chain



Partial Orders: Assymetry

Another way to describe a DAG is through the property of Assymetry:

Definition: Assymetric Relation

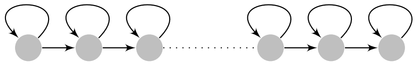
A relation **R** is assymetric if: $u\ R^+\ v \implies \mathrm{NOT}(v\ R^+\ u)$

In other words: a relation is assymetric if a path can only exist from u to v, or from v to u.

This makes sense, a DAG is characterized by the **lack of cycles**. If there are no cycles, it is not possible to go both ways.

Weak Partial Order

A weak partial order is a relaxation of the strict partial order. The difference is that each element is comparable to itself:



Examples:

- The $\leq$ relation $(1 \leq 1)$
- The $\subseteq$ relation $(\{1,2\} \subseteq \{1,2\})$

Weak partial orders help us define the Antisymmetry property:

Definition: Antissymetric Relation

A relation **R** is antissymetric if: $u\ R^+\ v \implies$ NOT$(v\ R^+\ u)$ ONLY WHEN $u \neq v$

Assimetry and Antissimetry

Assimetry

- Reflexibility is never allowed
- R is assimetric **iff**:

$$xRy \implies \text{NOT}(yRx)$$

Antissimetry

- Reflexibility is allowed ONLY WHEN $x = y$
- R is antissimetric **iff**

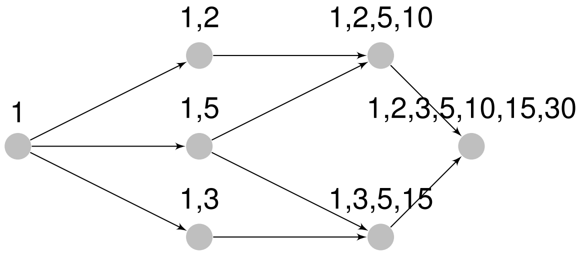
$$xRy \implies \text{NOT}(yRx), \text{ for } x \neq y$$

Proper Subset Relation

Proper Subset Relation

The proper subset relation: $A \subset B$ represents a partial order.

For example, the proper subset relation defined on the "recursive set of divisors of 30" is as follows:



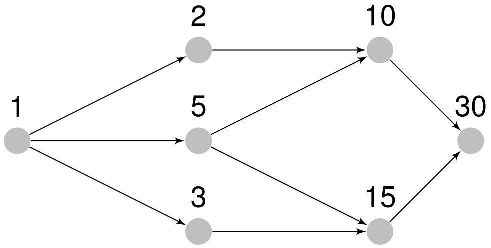
Partial Orders and Isomorphism

Proper Divide Relation

The proper divide relation is defined as aRb if $a|b$ and $a \neq b$.

We can see that, for the set $\{1, 2, 3, 5, 10, 15, 30\}$, the proper subset relation and the proper division relation have **the same relationship DAG.**

This means that the two relations are **isomorphic**



Isomorphism

- Two graphs are isomorphic if they have the same set of vertices and set of edges
- More formally, two graphs G_1, G_2 are isomorphic if there is a relation M which is an *edge preserving maching* between their vertices.
- G_1 isomorphic $G_2 \iff \exists$ bijection $M : V_1 \to V_2$
  with $(u, v) \in E_1 \iff (M(u), M(v)) \in E_2$

Symmetric Relations and Equivalence Relations

- If there is a walk from u to v and a walk from v to u, then we say that u and v are strongly connected.
    - uG^*v and vG^*u
- The relation R is symmetric if $aRb \implies bRa$.
    - The walk relation of A strongly connected graph is symmetric.
- An equivalence relation R is: transitive, symmetric and reflexive.
- This means that R is an equivalence relation **iff** R is the strongly connected relation of some DiGraph.

Equivalence Relations Examples

The definitions of the last slide allows us to formally define an *equivalence equation*.

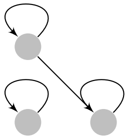
Examples:

- Equality: $=$
- $\equiv$(mod n)
- Same Size, Same Color, etc.

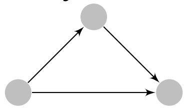
It may seem that an equivalence relation is too obvious to need a definition (specially for numbers!), but this can be useful when we want to define equivalence for more complex things, like sets.

Relation Properties: Graphical Review

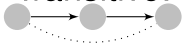
Reflexive:



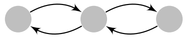
Assymetric:



Transitive:



Symetric:



Slide Credits

Image Credits I

1. Course Pre-requisite image from "Math for CS" MIT OCW