

GB13604 - Maths for Computer Science

Lecture 4 – Graphs Part I

Claus Aranha

caranha@cs.tsukuba.ac.jp

College of Information Science

2018-10-28

Last updated October 27, 2020

This course is based on Mathematics for Computer Science, Spring 2015, by Albert Meyer and Adam Chlipala, Massachusetts Institute of Technology OpenCourseWare.



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

① Graphs

② Walks

③ Scheduling and Partial Orders

Using Graphs to Solve Problems

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	<i>none</i>
0001	Intro to Programming	<i>none</i>
0002	Calculus I	<i>none</i>
0003	Programming Theory	<i>0001</i>
0004	Linear Algebra	<i>0000, 0002</i>
0005	Programming Challenges	<i>0000, 0001, 0003</i>
0006	Computer Graphics	<i>0003, 0004</i>

Using Graphs to Solve Problems

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?

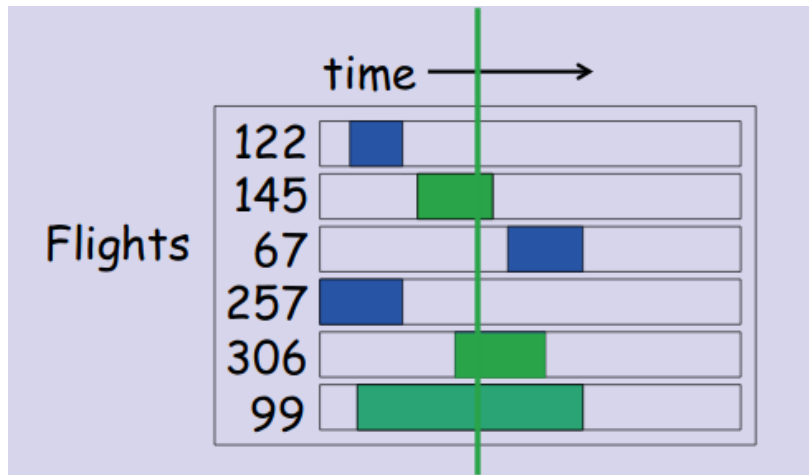
Code	Lecture	Prerequisites
0000	Social Questions	<i>none</i>
0001	Intro to Programming	<i>none</i>
0002	Calculus I	<i>none</i>
0003	Programming Theory	<i>0001, 0007</i>
0004	Linear Algebra	<i>0000, 0002</i>
0005	Programming Challenges	<i>0000, 0001, 0003</i>
0006	Computer Graphics	<i>0003, 0004</i>
0007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

Using Graphs to Solve Problems

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?



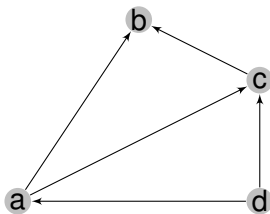
Problems as Graphs

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 Basic Definitions



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

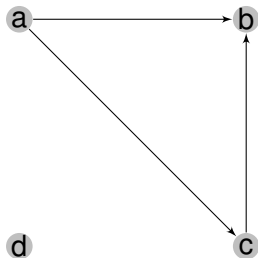
- $G = (V, E)$
- $V = \{a, b, c, d\}$
- $E = \{(a, b), (a, c), (c, b), (d, c), (d, a)\}$

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?



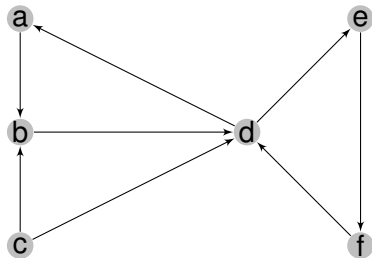
Graph $G = (V, E)$

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

A digraph with vertices V can be understood as a **binary relation** $V \rightarrow V$

In the same way, every **binary relation** can be written 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

Adjacency Matrix

An **Adjacency Matrix** A represents a digraph, when $A_{i,j}$ is 1 if $v_i \rightarrow v_j$, and 0 if not.

$$A(v_i, v_j) = 1 \iff E(v_i) = v_j$$

Part 2: Walks in Graphs

- 1 Graphs
- 2 Walks**
- 3 Scheduling and Partial Orders

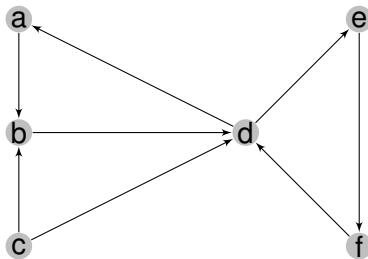
Walks and Paths

The example of the "course schedule" shows that the solution of a problem can be represented as a sequence of vertices or edges.

We can talk about these sequences as "Walks" or "Paths".

- A **Walk** is a sequence of successive edges;
- A **Path** is a walk that does not repeat vertices;

Walk example

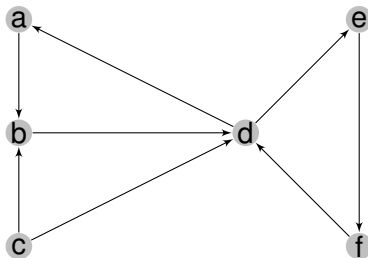


Walk: a sequence of successive edges

$b \rightarrow d \rightarrow e \rightarrow f \rightarrow d \rightarrow e$

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

Path example



Path: A walk without repeated vertices

e → f → d → a → b

Stuck!

- **Walk length:** 4 edges (Not 5 vertices)
- Can you find a walk that visits all vertices in the graph?

Walks and Paths

Simple example proof with graphs: *"The shortest walk between two vertices is a Path"*

Proof.

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 vertex v_k :

$$w_s = v_0 \rightarrow v_1 \rightarrow \dots \rightarrow v_k \rightarrow \dots \rightarrow v_k \rightarrow \dots \rightarrow v_{n-1} \rightarrow v_n$$

- 2 The walk w_s from v_0 to v_n , contains a small walk w_k from v_k to v_k of size $|w_k| \geq 1$.
- 3 We can remove w_k from w_s , resulting in a smaller walk w'_s with size $|w_s| - |w_k|$
- 4 w'_s is a walk from v_0 to v_n that is shorter than w_s , which is a contradiction.



The Walk Relation

We define the **Length n Walk Relation**:

$$vG^n w. \quad (1)$$

This means "There is a walk of length exactly n from v to w in G ".

Some facts about the G^n

- G^1 is the relation of vertices directly connected by edges in G .
- **addition:** $G^n \circ G^m = G^{n+m}$ (the composite of two Walk Relations is their addition)
- **common vertex:** $x G^m \circ G^n y \rightarrow \exists z, x G^m z G^n y$ ($G^m \circ G^n$ has a common vertex)

Walk Relations: Composition and the Adjacency Matrix

Consider the adjacency matrix A_{G^n} that describes the relation G^n : $a_{ij} = 1 \iff v_i G v_j$.

It is possible to calculate the Adjacency Matrix of a composite walk relation using *boolean matrix multiplication*: $A_{G^n \circ G^m} = A'_{G^n} \odot A''_{G^m}$:

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

This means that we can calculate A_{G^n} quickly using the [Fast Matrix Exponentiation](#) algorithm:

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

Walk Relation G^* of a Digraph


- G^n is the length n walk relation; while G^* is the walk relation of G
- uG^*v means that there is a walk from u to v of any length

QUIZ: How do we calculate G^* for a graph, given the adjacency matrix A ?

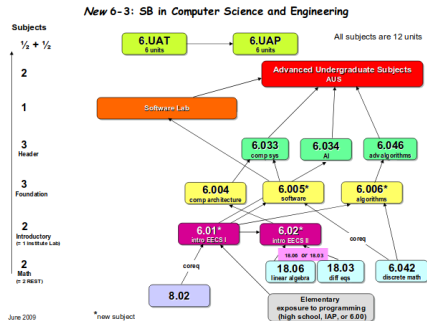
Remember that a walk can be infinite. So is it possible to solve this problem with a **finite** algorithm?

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:  (Or the identity matrix).
- 3 The relation $G^{\leq 1} = G^0 \cup G^1$ is the relation for walks of *length 1 or less*;
- 4 $G^* = (G^{\leq})^{n-1}$, so you can calculate G^* in $\log n$ matrix boolean multiplications

Walks and Prerequisite Course Graphs



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

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

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.
 - $v_0 \rightarrow v_1 \rightarrow \dots \rightarrow v_n \rightarrow v_0 \mid i > 0, j > 0, v_i \neq v_j$
 - 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

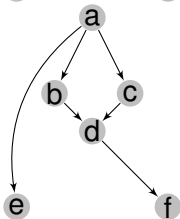
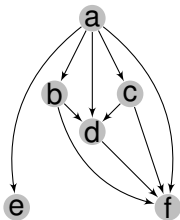
Directed Acyclic Graphs (DAGs), can be used to represent several ordered structures:

- 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: $n \rightarrow n + 1$
 - Subset Relation: $\{1, 2\} \subset \{1, 2, 3\}$
- Induction Proofs: $(P(n) \implies P(n + 1) \implies P(n + 2) \dots)$;
- 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

- ① Graphs
- ② Walks
- 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

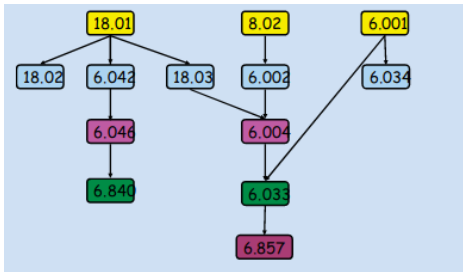
18.03, 6.002 \rightarrow 6.004

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

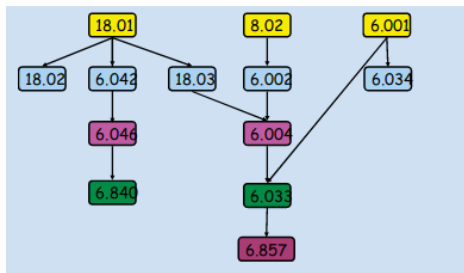
Minimal, Minimum, Maximal, Maximum of a DAG



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

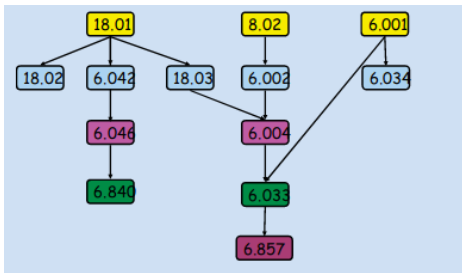
- ① Identify Minimal Subjects;
- ② Add Minimal Subjects to Schedule;
- ③ Remove Minimal Subjects;
- ④ Return to Step 1

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

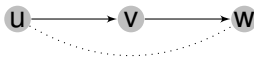
Parallel Processing

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: 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



Representing this in terms of **walk relation** in G :

$$uG^+v \wedge vG^+w \implies uG^+w$$

Definition: Transitive Relations

A relation \mathbf{R} is transitive if: $xRy \wedge yRz \implies xRz$

Partial Orders: Assymetry

In an **acyclic digraph** G , we can observe that for any two vertices v and u , if there is a walk from v to u , then there is no walk from u to v .

Definition: Assimetric Relation

A relation \mathbf{R} is assimetric if: $uD^+v \implies \text{NOT}(vD^+u)$

Strict Partial Order

A relation R is a **Strict Partial Order**(SPO) **iff** it is **Transitive** and **Assimetric**.

Examples:

- The \subset relation on sets
- The “indirect prerequisite” relationship on subjects.
- The $<$ relationship on \mathbb{R}

Another way to say it, is that R is a **SPO** **iff** R is the walk relation D^+ for some DAG D .

Path Total Orders

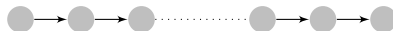
A **Strict Partial Order** is also **Path Total** if, for any two distinct elements, one will always be “greater than” another.

Example: $<$ on \mathbb{R} : if $x, y \in \mathbb{R}, x \neq y \implies x > y$ or $y > x$

Counter-Example: \subset in $\text{POW}(\mathbb{N})$: $\{1, 3\} \not\subset \{2, 5\} \not\subset \{1, 3\}$

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

In a **path total** relation, the whole graph is a **chain**



Weak Partial Order

A **weak partial order** is the same as a **strict partial order** R , except that aRa always holds:



- Examples: \subseteq on sets, \leq on \mathbb{R}
- Weak Partial Orders define the property of **Reflexivity**
- Relation R on A is **reflexive** **iff** $aRa, \forall a \in A$

Another way to define a weak partial order is that R is a WPO **iff** $R = D^*$ for some DAG D

Assimetry and Antissimetry

Assimetry

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

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

Antissimetry

- Reflexibility is **sometimes** allowed
- R is **antissimetric iff**

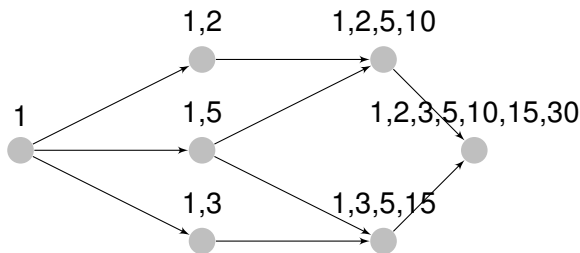
$$xRy \implies \text{NOT}(yRx), \text{ for } x \neq y$$

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 power set of $\{1, 2, 3, 5, 10, 15, 30\} - \emptyset$ is as follows:



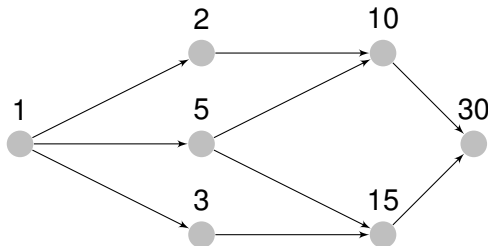
Partial Orders and Isomorphism

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 matching* between their vertices.
- G_1 isomorphic $G_2 \iff \exists$ bijection $M : V_1 \rightarrow V_2$
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*.

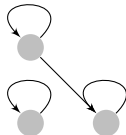
Examples:

- Equality: $=$
- $\equiv (\text{mod } n)$
- 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

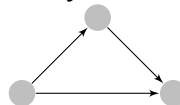
Reflexive:



Transitive:



Assymetric:



Symetric:



Slide Credits

These slides were made by Claus Aranha, 2020. You are welcome to copy, re-use and modify this material, following the CC-SA-NC license.

These slides are based on "Mathematics For Computer Science (Spring 2015)", by Albert Meyer and Adam Chlipala, MIT OpenCourseWare. <https://ocw.mit.edu>.

Individual images in some slides might have been made by other authors. Please see the following slides for information about these cases.

Image Credits I

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