Graph Theory

Daniel Krashen

February 7, 2016

Contents

Version Notes **3**
- 0.1 Updates 1/28/2016 . . . 3
- 0.2 Updates 1/30/2016 . . . 3

1 Lecture 1: The language of graphs **4**
- 1.1 Graphs . . . 4
- 1.2 Real world graph problems . . . 6
  - 1.2.1 Scheduling . . . 6
  - 1.2.2 Tournaments . . . 6
- 1.3 The rationale behind the language . . . 7
- 1.4 Other basic notions . . . 7
- 1.5 Exercises . . . 7

2 Lecture 2: Digraphs and degree formulas **8**
- 2.1 Directed graphs . . . 8
- 2.2 From graphs to digraphs . . . 9
- 2.3 Exercises . . . 10

3 Lecture 3: Subgraphs, isomorphisms **11**
- 3.1 Isomorphisms . . . 11
  - 3.1.1 Simplifications for simple graphs . . . 11
- 3.2 Subgraphs . . . 12
- 3.3 Unions, Intersections . . . 13
- 3.4 Other operations on graphs . . . 14
- 3.5 Exercises . . . 14

4 Lecture 4: Paths, walks, trails, circuits, cycles **15**
- 4.1 Walks and connectedness . . . 15
  - 4.1.1 Bridges . . . 16
- 4.2 Trails, paths, cycles, circuits . . . 17
- 4.3 Exercises . . . 17

5 Lecture 5: Trees and spanning trees **18**
5.1 Trees 18
5.2 Dijkstra's algorithm 19
5.2.1 The algorithm 20

6 Lecture 6: More Trees **22**
6.1 Spanning trees and cycles 22

7 Lecture 7: Connectivity, part 1 **23**
7.1 Different notions of (dis-)connectivity 23

8 Lecture 8: More Connectivity **25**
8.1 Bonds, cotrees and cuts 25

9 Tours **26**
9.1 Regular graphs 26

A Foundational notions **27**
A.1 Sets and multisets 27
A.2 Relations 28
A.2.1 Functions 28
A.2.2 Equivalence Relations 28

Version Notes

0.1 Updates 1/28/2016

- added Section 1.4, to introduce some additional notation
- added spanning subgraph material (some) in Section 5

0.2 Updates 1/30/2016

- added Lemma 5.2.3 on the optimality of Dijkstra's algorithm.
- reorganized exercises (to ends of chapters)

Chapter 1

Lecture 1: The language of graphs

1.1 Graphs

Graphs encode the idea of connections between things, for example

- networks of computers
- people and their relationships
- cities and highways
- sets and intersections
- workers and tasks

In formal mathematical terms, a graph is:

Definition 1.1.1. *A **graph** G is an ordered triple (V, E, ψ) consisting of*

- *a finite, nonempty set V, whose elements are referred to as vertices,*
- *a finite (possibly empty) set E, whose elements are referred to as edges, and*
- *an "incidence" function $\psi : E \to \mathscr{R}_2(V)$,*

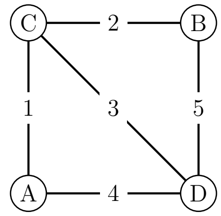
where $\mathscr{R}_2(V)$ is the set of unordered pairs of elements of V (which one may also think of as two elements multisubsets of V – see Definition A.1.10).

We represent such graphs visually by drawing vertices as dots or circles, and edges as lines between them. For example, the drawing in Figure 1.1 would stand for the graph G defined by $V_G = \{A, B, C, D\}, E_G = \{1, 2, 3, 4, 5\}$ and incidence function ψ_G given by:

$$\psi_G(1) = AC,\ \psi_G(2) = BC,\ \psi_G(3) = CD,\ \psi_G(4) = AD,\ \psi_G(5) = BD.$$

Notation 1.1.2. *For a graph $G = (V, E, \psi)$ we write V_G for V, E_G for E and ψ_G for ψ.*

Figure 1.1:



In other words, using this notational convention, if we are given graphs G, H, K, and have not specified letters for their sets of vertices, edges, etcetera, we may write, for example, E_K for the edges of the graph K, V_H for the vertices of H, and ψ_G for the incidence function of G.

Definition 1.1.3. *A graph is called* ***trivial*** *if it has a single vertex (see Figure* **??***).*

Definition 1.1.4. *Let G be a graph, $e \in E_G$ an edge and $v \in V_G$ a vertex. We say that e and v are* ***incident*** *if $v \in \psi(e)$.*

Definition 1.1.5. *Let G be a graph $v, w \in V_G$. We say that v and w are* ***adjacent*** *if there is an edge e with v and w incident to e.*

Definition 1.1.6. *Let G be a graph $e, e' \in E_G$. We say that e and e' are* ***adjacent*** *if there is a vertex v with e and e' incident to v.*

DIAGRAM

Definition 1.1.7. *Let G be a graph. If $e \in E_G$ is an edge, we say that e is a* ***loop****, if e is incident to exactly one vertex.*

Definition 1.1.8. *We say that G is a* ***simple graph*** *if*

- *G has no loops,*
- *there is at most 1 edge incident to any pair of vertices.*

Note that the second condition is the same as requiring that the function ψ_G be one-to-one.

Graphs can be drawn in many different ways:

Definition 1.1.9. *G is called a* ***planar graph*** *if it may be drawn in the plane with no edges crossing.*

1.2 Real world graph problems

1.2.1 Scheduling

- vertices = jobs that need to be done
- edges = jobs which require conflicting resources

problem: how to decide how many "periods of work" needed to complete all jobs.

Similar problem: table arrangements at a wedding

- vertices = guests
- edges = guest that don't get along

problem: how many tables?

translation: vertex colorings, chromatic number of a graph

Definition 1.2.1. *A* ***vertex coloring*** *of a graph G is a function f defined on the set of vertices of G. An vertex n-coloring is a function $f : V_G \to \{1, \ldots, n\}$. We say that a vertex coloring is* ***proper*** *if $f(v) \neq f(w)$ for adjacent vertices v, w.*

Definition 1.2.2. *The chromatic number of a graph, $\chi(G)$, is the minimal number n such that G has a proper vertex n-coloring.*

1.2.2 Tournaments

various teams need to play each other. disjoint pairs of teams can play simultaneously, but of course the same team can't play at the same time. How many rounds are needed for teams to play each other?

- vertices = teams
- edges = teams who need to play each other

problem: how many rounds?

Definition 1.2.3. *An* ***edge coloring*** *of a graph G is a function f defined on the set of edges of G. An edge n-coloring is a function $f : E_G \to \{1, \ldots, n\}$. We say that an edge coloring is* ***proper****proper!edge coloring if $f(e) \neq f(e')$ for adjacent vertices e, e'.*

Definition 1.2.4. *The* ***edge chromatic number*** *of a graph, $\chi'(G)$, is the minimal number n such that G has a proper edge n-coloring.*

1.3 The rationale behind the language

The choice of thinking of a graph as a triple $G = (V, E, \psi)$ has its advantages and disadvantages. If we were only concerned with simple graphs, we could have simplified our notation somewhat by omitting the function ψ, and letting E itself be a subset of the set of unordered pairs of distinct elements of V. In the case of general graphs, however, where there can be multiple edges between two vertices, this is somewhat less convenient. We could persist with this approach by saying that E be a multisubset instead of a subset, however, this is a little bit less convenient later when we wish to talk about colorings or labellings of edges.

An alternate way of defining things could be as follows: Instead of defining the function ψ as the fundamental concept, one may instead define the notion of **incidence** as the fundamental concept as follows:

Definition 1.3.1. *A griph G is an ordered triple (V, E, α) consisting of a set of vertices V, a set of edges E, and a set of ordered pairs $\alpha \subset V \times E$ such that*

for every $e \in E$, there is at least one, and at most two elements $v \in V$ such that $(v, e) \in \alpha$. If $(v, e) \in \alpha$, we say that v is incident to e. A griph is called simple if every edge is incident to exactly two vertices.

1.4 Other basic notions

For a graph G, we let $v(G) = \#V_G$ denote the number of vertices of G, and $e(G) = \#E_G$ denote the number of edges of G. We let $\delta(G)$ denote the mininum degree of a vertex of G, and $\Delta(G)$ denote the maximum degree of a vertex.

1.5 Exercises

Exercise 1.5.1. *Show that griphs are exactly in correspondence with graphs in such a way that the relationship of incidence lines up.*

Chapter 2

Lecture 2: Digraphs and degree formulas

2.1 Directed graphs

A variation on the notion of a graph is also very useful both theoretically and in applications:

Definition 2.1.1. *A* ***directed graph*** *or* ***digraph*** *D is an ordered triple (V, A, ψ) where V is a set, referred to as the* ***vertices*** *of D, a set A referred to as the* ***arrows*** *of D, and a pair of functions $s, t : A \to V$, taking arrows to elements of V.*

Notation 2.1.2. *For a digraph $D = (V, A, \psi)$, as before, we write V_D for V, A_D for A, s_D for s, and t_D for t.*

Notation 2.1.3. *For a digraph D, and an arrow $a \in A_D$, we call $s(a)$ the* ***source*** *of a and $t(a)$ the* ***target*** *of a.*

EXAMPLES:

- one way street maps
- irreversible processes
- dependencies: e.g. scheduling with dependencies

Definition 2.1.4. *Let D be a digraph. For a vertex v, we define the* ***outdegree*** *of v, denoted* $\operatorname{outdeg} v$*, to be the number of edges whose source is v, and the* ***indegree*** *of v, denoted* $\operatorname{indeg} v$*, to be the number of edges whose target is v. Formall, we have*

$$\operatorname{outdeg} v = \#\{a \in A_D \mid s(a) = v\}, \quad \operatorname{indeg} v = \#\{a \in A_D \mid t(a) = v\}.$$

If it is necessary to specify the digraph, we may also write $\operatorname{outdeg}_D(v)$ *or* $\operatorname{indeg}_D(v)$.

Proposition 2.1.5 (The degree formula for digraphs)**.** *Suppose that D is a digraph. Then*

$$\sum_{v \in V_D} \operatorname{outdeg}(v) = \sum_{v \in V_D} \operatorname{indeg}(v) = \# A_D.$$

Proof. Informally, this is clear for the following reason: every arrow in A_D has its source at exactly one vertex, and so contributes exactly 1 to the first sum, and every arrow has its target at exactly one vertex and similarly contributes exactly once to the second sum.

Let us, however, for the sake of practice, give a more formal argument:

$$\sum_{v \in V_D} \operatorname{outdeg}(v) = \sum_{v \in V_D} \sum_{a \in A_D, s(a)=v} 1 = \sum_{(v,a) \in V_D \times A_D, s(a)=v} 1$$

But now, let us notice that the pairs (v, a) with $s(a) = v$ are in bijection with simply the set A_D, since by the description, v is determined by a. Therefore, we may rewrite this as:

$$\sum_{(v,a) \in V_D \times A_D, s(a)=v} 1 = \sum_{a \in A_D} 1 = \# A_D.$$

The rest of the proof follows in an analogous way.

□

2.2 From graphs to digraphs

Given a graph G, we may construct a digraph $dig(G)$ by definining

- $V_{dig(G)} = V_G$,
- $A_{dig(G)} = \{(v, e) \in V \times E | v \in \psi(e)\}$,
- $s_{dig(G)}(v, e) = v$, $t_{dig(G)}(v, e) = w$, where $\psi(e) = vw$.

Definition 2.2.1. *Suppose that G is a graph. We define the* **degree** *of a vertex $v \in V_G$, denoted* $\deg v$ *to be the number of edges incident to v. If it is necessary to specify the graph, we may also write* $\deg_G v$*.*

Proposition 2.2.2 (The degree formula for graphs)**.** *Suppose that G is any graph. Then we have*

$$\sum_{v \in V_G} \deg v = 2\# E.$$

Proof. Intuitively, one way to see this is that if we chop each edge in the middle, making two "half edges" for every edge, then each half edge is incident to exactly one vertex, and contritutes exactly once to the degree count on the left.

More formally, if we let $dig(G)$ be the associated digraph to G, then we note that for every edge of G, there are two arrows of $dig(G)$. Also, for every edge e incident to v, there is exactly one arrow, which we call (v, e) which starts at e. That is, the map

$$\{e \in E_G | e \text{ is incident to } v\} \to \{a \in A_{dig(G)} | s(a) = v\}$$
$$e \mapsto (v, e)$$

is a bijection (its inverse being given by $(v, e) \mapsto e$). In particular, this says that $\operatorname{outdeg}_{dig(G)} v = \deg_G v$.

By the degree formula for digraphs, we therefore have

$$\sum_{v \in V_G} \deg_G v = \sum_{v \in A_{dig(G)}} \operatorname{outdeg}_{dig(G)} v = \#A_{dig(G)} = 2\#E_G$$

as desired. $\square$

A surprising conclusion here is that the sum of the degrees of the vertices of a graph must be even!

2.3 Exercises

Chapter 3

Lecture 3: Subgraphs, isomorphisms

3.1 Isomorphisms

Definition 3.1.1. *Given two graphs G, H, an **isomorphism** f from G to H, written $f : G \to H$ is a pair of maps $f = (f_V, f_E)$, where $f_V : V_G \to V_H$ is a function from the vertices of G to the vertices of H and $f_E : E_G \to E_H$ is a function from the edges of G to the edges of H such that both f_V and f_E are bijective and such that for $v \in V_G$, $e \in E_G$, we have that v and e are incident if and only if $f_V(v)$ is incident to $f_E(e)$.*

Notation 3.1.2. *If G and H are graphs, we write $G \cong H$ and say that G and H are **isomorphic** if there exists an isomorphism $f : G \to H$.*

In other words, thinking of the sets V_G, E_G as the labels for the vertices and edges of G and thinking of V_H, E_H as the labels for the vertices and edges of H, we may think of f_V and f_E as re-assigning the labels of the vertices and edges of a graph. The final property says that the relation of incidence is preserved. Note that the incidence relation encodes the function ψ, as v and e are incident if and only if $v \in \psi(e)$ (and since the unordered pair/two element multiset $\psi(e)$ is determined precisely by which elements it contains).

This is very useful, as our main concern is structural information about graphs, as opposed to the specific names which we have assigned to their edges and vertices. For this reason, when drawing a graph, we will typically omit names for the vertices and edges, drawing our graphs as in figure 3.1 instead.

3.1.1 Simplifications for simple graphs

It is perhaps useful to note that this definition may be made a bit simpler in the case of simple graphs (not suprising, I'm sure). To be precise:

Expanding on this idea a bit, we see that in a simple graph, the graph is determined up to isomorphism by the relationship of adjacency – that is, which vertices are joined by an edge. That is to say, although these two graphs shown in Figure 3.1.1 are different as graphs, they are still isomorphic canonical way, via the labelling of the vertices.

Figure 3.1: some graphs

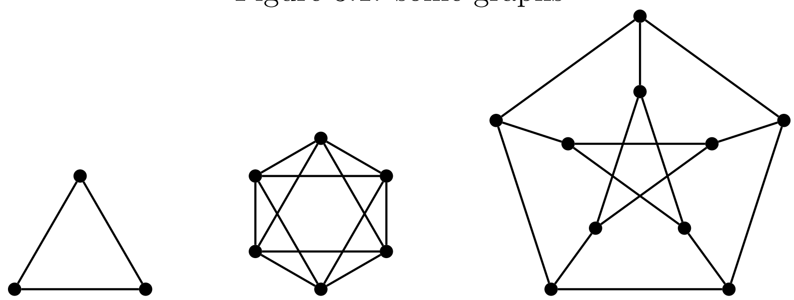
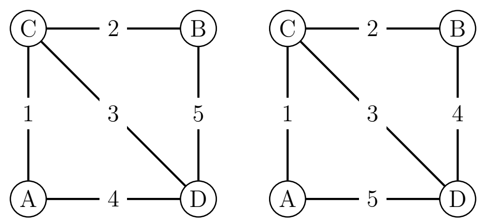


Figure 3.2: isomorphic graphs



Notation 3.1.3. *Along the same lines, for a simple graph $G = (V, E, \psi)$, if e is an edge with $\psi(e) = vw$, we will abuse language and refer to vw as e. In other words, the statement that for a pair of vertices $v, w \in V$, $vw \in E$ means that there is some edge e with $\psi(e) = vw$, or equivalently v and w are adjacent.*

Definition 3.1.4. *For a simple graph G, we define the* ***complement*** *G, denoted $\overline{G}$ to be the graph with the same set of vertices (i.e. $V_G = V_{\overline{G}}$) and such that a pair of vertices v, w are adjacent in $\overline{G}$ if and only if they are not adjacent in G.*

3.2 Subgraphs

Definition 3.2.1. *Let G be a graph. We say that a graph H is a* ***subgraph*** *of a graph G if $V_H \subset V_G$, $E_H \subset E_G$ and $\psi_H = \psi_G|_{E_H}$. If H is a subgraph of G, we write $H \subset G$.*

Definition 3.2.2. *Let G be a graph, and $W \subset V_G$ a subset of its vertices. We define a new graph $G[W]$, called* ***the subgraph of*** *G* ***induced by*** *W, to be the graph whose vertex set is W and whose edge set $E_{G[W]}$ consists of all the edges of G which are only incident to vertices in W, together with the same incidence relations. Formally, we set*

$$V_{G[W]} = W,\ E_{G[W]} = \{e \in E_G \mid \psi(e) \subset W\},\ \psi_{G[W]} = \psi|_{E_{G[W]}},$$

where $\psi|_{E_{G[W]}}$ denotes the restriction of the function ψ to the edges of $G[W]$.

Given a graph G, we will often want to understand its structure by looking at which subgraphs it has, and their properties. For example, consider the following famous statement:

> In every group of 6 people, there is either a group of 3 people all of whom know each other, or a group of 3 people, none of whom know each other.

One convenient way of conceptualizing this question within the framework of graph theory is as follows: consider the two special graphs below

TRIANGLE DISCRETE GRAPH WITH 3 VERTICES

We may now formally state the previous result as follows:

In generalizing this result, which we will look into later in Chapter **??**, it is natural to want to consider groups of n vertices in a graph, all of which are connected. The correponding subgraphs of interest are called the complete graphs:

Definition 3.2.3. *The* ***complete graph*** *on n vertices, denoted K_n is the simple graph with vertices consisting of the set $\{1, \dots, n\}$, and where every two vertices are adjacent.*

Definition 3.2.4. *Let G be a simple graph. An n-**clique** in G is a collection of vertices $v_1, \dots, v_n \in V_G$ such that the induced subgraph $G[\{v_1, \dots, v_n\}]$ is isomorphic to K_n.*

It is very natural to ask, for a given graph, about the existence or non-existence of cliques of a given size – see, for example exercise .

Another way to generalize the graph K_3 is in the notion of a cycle graph:

Definition 3.2.5. *The* ***cycle graph*** *on n vertices ($n \geq 3$)[1], denoted C_n is the simple graph with vertices consisting of the set $\{1, \dots, n\}$, and where vertices i and j are adjacent if and only if $|i - j|$ is 1 or $n - 1$.*

In other words, there are edges between vertices which are 1 unit apart, and one additional edge connecting 1 and n.

Definition 3.2.6. *A cycle in a (not necessarily simple) graph G is a subgraph $C \subset G$ such that $C \cong C_n$ for some $n \geq 3$.*

As we will see, cycles and cliques are interesting for a variety of reasons, both practically and theoretically. Intuitively, one should view the existence of cycles and cliques as helping to describe how highly connected a graph is, somehow encapsulating "redundancies of connections." We will explore these ideas more in Chapters **??**.

3.3 Unions, Intersections

Definition 3.3.1. *Let G be a graph and H_1, H_2 sugraphs of G. We say that G is the* ***union*** *of H_1 and H_2, and write $G = H_1 \cup H_2$ if $V_G = V_{H_1} \cup V_{H_2}$ and $E_G = E_{H_1} \cup E_{H_2}$. We say that the union is* ***disjoint*** *if $V_{H_1} \cap V_{H_2} = \emptyset$, and we say that the union is* ***edge disjoint*** *if $E_{H_1} \cap E_{H_2} = \emptyset$.*

[1] we let bigons be bigons, as they say

Note that a disjoint union is automatically edge disjoint as well, since if two subgraphs share a common edge it must also share any vertices which are incident to it. Note also that whenever we have any two subgraphs H_1, H_2 in a graph G, we may find a unique subgraph $H < G$ with $H = H_1 \cup H_2$.

Definition 3.3.2. *Let G be a graph and $H_1, H_2 < G$ subgraphs of G. We define the **intersection** $H_1 \cap H_2$ to be the subgraph with $V_{H_1 \cap H_2} = V_{H_1} \cap V_{H_2}$ and $E_{H_1 \cap H_2} = E_{H_1} \cap E_{H_2}$.*

3.4 Other operations on graphs

Definition 3.4.1. *Let G be a graph, and $e \in E_G$. We define a subgraph $G - e$ of G by $V_{G-e} = V_G$ and $E_{G-e} = E_G \setminus \{e\}$.*

Definition 3.4.2. *Let G be a graph, and $e \in E_G$. If $H < G$ is a subgraph, we define the new subgraph $H + e$ to be the subgraph with*

$$V_{H+e} = V_H \cup \{v \in G |\ v \textit{ incident to } e\}$$

and

$$E_{H+e} = E_H \cup \{e\}$$

Definition 3.4.3. *Let G be a graph, and $v \in V_G$. We define the subgraph $G - v$ of G to be the induced subgraph $G - v = G[V \setminus \{v\}$. That is, it is the graph obtained by removing the vertex v as well as all edges incident to v.*

3.5 Exercises

Exercise 3.5.1. *Suppose that G and H are simple graphs. Suppose we have a bijective function $g : V_G \to V_H$. Then there exists a unique graph isomorphism $f = (f_V, f_E) : G \to E$ with $f_V = g$ if and only if for every two vertices $v, w \in V_G$, we have that v is adjacent to w if and only if $g(v)$ is adjacent to $g(w)$.*

Exercise 3.5.2. *Show that every simple graph with 6 vertices must contain an induced subgraph isomorphic to one of the above graphs[2].*

Exercise 3.5.3. *Find the smallest number n such that every simple graph with n edges and 6 vertices has a 3-clique.*

Exercise 3.5.4. *Give an example of a simple graph with 4 vertices and exactly 3 cycles, and a graph with 3 vertices and exactly 2 cycles.*

Exercise 3.5.5. *Verify that definition 3.3.2 does in fact give a well-defined subgraph.*

[2]hint: as a first step, considering the friends of one particular person A, partitions the other 5 people into two groups (friends of A and not friends of A), it follows that one of these two groups must have at least 3 people

Chapter 4

Lecture 4: Paths, walks, trails, circuits, cycles

4.1 Walks and connectedness

This chapter will introduce a fair amount of language which will be useful in subsequent lectures.

Definition 4.1.1. *A **walk** W in a graph G is a sequence of alternating vertices and edges*

$$W = v_1 e_2 v_2 e_3 v_3 \cdots e_n v_n$$

*where $v_1, \ldots, v_n \in V_G$, $e_2, \ldots, e_n \in E_G$, and such that e_i is incident to the vertices v_{i-1} and v_i. The vertex v_1 is called the **origin** and v_n is called the **terminus**.*

Convention 4.1.2. *We will allow a walk to have only a single vertex. That is, the sequence consisting only of a single vertex v is a valid walk with origin and terminus v. We will not allow a walk to be empty, however.*

Notation 4.1.3. *It is easy to see that if G is a simple graph, a walk $W = v_1 e_2 \cdots e_n v_n$ is completely determined by its list of vertices. For this reason, if G is simple, we will generally write $W = v_1 v_2 \cdots v_n$ for convenience.*

Definition 4.1.4. *Let G be a graph, $v, w \in V_G$. A (v, w)-walk is a walk with origin v and terminus w.*

Definition 4.1.5. *Let G be a graph, $W = v_1 e_2 \cdots e_n v_n$ and $W' = v'_1 e'_2 \cdots e'_m v'_m$ walks with $v_n = v'_1$. We define the concatenation WW' of the two walks to be the (v_1, v'_m) walk defined by the sequence*

$$v_1 e_2 \cdots e_n v_n e'_2 v'_2 \cdots e'_m v_m$$

Definition 4.1.6. *Let G be a graph, and $W = v_1 e_2 \cdots e_n v_n$ a walk. We define the reverse of W, denoted W^{-1} to be the walk $v_n e_n v_{n-1} \cdots e_2 v_1$.*

In Exercise 4.3.1, we defined an equivalence relation on the set V_G of vertices of G, by saying that two vertices v, w are equivalent if there is a (v, w)-walk in G.

In this way, we find that the set V_G of vertices of G can be written as a disjoint union of equivalence classes $V_G = \bigcup_{i=1}^{n} V_i$.

Definition 4.1.7. *The induced subgraphs $G[V_i]$ are called the* ***components*** *of G. The number n of components of G is denoted $c(G)$. We say that a graph G is* ***connected*** *if it has a single component.*

Note that a graph is a disjoint union of its components, $G = \bigcup_{i=1}^{c(G)} G[V_i]$.

4.1.1 Bridges

Definition 4.1.8. *We say that $e \in E_G$ is a bridge if $c(G - e) > c(G)$.*

Lemma 4.1.9. *Suppose that e is a bridge. Then $c(G - e) = c(G) + 1$.*

Proof. Let us begin with the case that $c(G) = 1$. It is easy to see that if e is a bridge, then it cannot be a loop. In paticular, it is incident to two distinct vertices v_1, v_2. It is also clear that e must be the unique edge connecting its two incident vertices. Let G_1 be the connected component of v_1 in $G - e$ and G_2 be the connected component of G_2 in $G - e$. We claim that $G - e$ is the disjoint union of G_1 and G_2, which would prove that $c(G) = 2$ as desired.

First, we note that G_1 and G_2 are disjoint, since they consist of vertices from distinct equivalence clases. Therefore, we need only to show that every vertex and edge of $G - e$ is either in G_1 or G_2.

Let us begin with the vertices. Let $w \in V_G = V_{G-e}$. By Lemma **??**, since G is connected, we can find a (w, v_1)-path in G.

If v_2 arises in this path, then looking at the first part of the path up to v_2 gives a (w, v_2)-path in G which does not involve the vertex v_1. In particular, the edge e cannot arise in this path. Therefore, this path is actually a (w, v_2)-path within the connected component of v_2 in $G - e$. That is, we have shown that $w \in V_2$.

On the other hand, if v_2 doesn't arise in this path, the (w, v_1)-path cannot involve the edge e, and in particular, shows that w is in V_1.

Finally, we check that the every edge of $G - e$ is either in G_1 or G_2. Given such an edge $e' \in E_{G-e}$, suppose v is incident to e'. By the above, v is in G_1 or G_2. Let us suppose that $v \in V_{G_1}$. Let w be the other (not necessarily distinct) vertex incident to e. We cannot have $w \in V_{G_2}$ since in this case we would have $w \sim v$, but by definition, they are in different components. Therefore both of the incident vertices for e' lie in the same component G_1, and by definition of the induced subgraph, the edge e' is in G_1 as well.

To complete the proof, consider the case that G may not be connected. In this case, write $G = \bigcup_{i=1}^{c(G)} G_i$ as a disjoint union of its components (as in Exercise 4.3.3). Then the edge e lies in some particular component, say $e \in G_1$. We have a disjoint union:

$$G - e = (G_1 - e) \cup \bigcup_{i=2}^{c(G)} G_i,$$

and by the above, either $c(G_1 - e)$ is 1 or 2. If it is 1, then $G_1 - e$ is connected, and $c(G-e) = c(G)$ by Exercise 4.3.3. If it is 2, then again by Exercise 4.3.3, we have $c(G-e) = c(G) + 1$. □

4.2 Trails, paths, cycles, circuits

Definition 4.2.1. *A walk $W = v_1e_2v_2 \cdots e_nv_n$ in a graph G is called a* ***trail*** *if all the edges e_i are distinct. It is called a* ***path*** *if all the vertices v_i are distinct.*

Note that paths are also nessarily trails.

Lemma 4.2.2. *Let v, w be vertices of a graph G. Suppose that there is a (v, w)-walk in G. Then there is a (v, w)-path in G.*

Proof. Suppose that $W = v_1e_2v_2 \cdots e_nv_n$ is a walk with $v_1 = v$ and $v_n = w$. Choose W of minimal length. We claim that W is a path. Supposing it is not, we would have $v_i = v_j$ for some $i < j$. But in this case, we may consider the new walk $W' = v_1e_2 \cdots e_{i-1}v_{i-1}v_ie_{j+1}v_{j+1} \cdots e_nv_n$. By assumption, this is a shorter walk, contradicting the minimality of W. Therefore W is a path as claimed. □

Definition 4.2.3. *A walk is called* ***closed*** *is its origin and terminus coincide. A closed walk $W = v_1e_2v_2 \cdots e_nv_ne_{n+1}v_1$ is called a* ***circuit*** *when $v_1e_2v_2 \cdots v_n$ is a trail, and it is called a* ***cycle*** *when $v_1e_2v_2 \cdots v_n$ is a path.*

Note here that there is some notational ambiguity here, as we have defined a cycle to both be a subgraph isomorphic to C_n, as well as a particular type of walk. Of course the connection is that if $v_1e_2v_2 \cdots e_{n+1}v_1$ is a cycle, then the vertices $v_1, v_2, \ldots, v_n$ together with the edges $e_2, e_3, \ldots, e_{n+1}$ give a subgraph isomorphic to C_n.

4.3 Exercises

Exercise 4.3.1. *Define a relation on the vertices of a graph G by saying that $v \sim w$ if and only if there exists a (v, w)-walk in G. Show that this gives an equivalence relation.*

Exercise 4.3.2. *Show that G is connected if and only if we cannot find nonempty subgraphs H_1, H_2 such that G is a disjoint union of H_1 and H_2.*

Exercise 4.3.3. *Somewhat more generally, show that*

1. *$c(G)$ is the maximum number such that we may write G as a disjoint union $G = \cup_{i=1}^{c(G)} G_i$.*

2. *If we have $G = \cup_{i=1}^{n} G_i$ a disjoint union, where each subgraph G_i is connected, then the G_i are the connected components of G and $n = c(G)$.*

Exercise 4.3.4. *Suppose G is a graph and $v \in V_G$ with $\deg(v) = 1$. Then $c(G) = c(G - v)$.*

Chapter 5

Lecture 5: Trees and spanning trees

Definition 5.0.1. *Let H be a subgraph of G. We say that H is a* ***spanning subgraph*** *if $V_H = V_G$.*

In this lecture we will consider spanning subgraphs of various types. These play a very important role for a variety of reasons.

5.1 Trees

Definition 5.1.1. *We say that a simple graph G is a* ***forest****, if it has no cycles; that is, if it has no subgraphs which are isomorphic to C_n, $n \geq 3$.*

Definition 5.1.2. *A* ***tree*** *is a connected forest.*

Definition 5.1.3. *A vertex of degree 1 in a tree is called a* ***leaf****.*

Lemma 5.1.4. *A graph G is a tree if and only if there is a unique (v, w)-path between any two vertices.*

Lemma 5.1.5. *If G is a tree then $e(G) + 1 = v(G)$.*

We will later see that the converse of this statement is true as well for a connected graph.

Proof. Let us first show that this equality holds for all trees. Assuming first that it doesn't, we may find a tree T with a minimum number of edges such that the equality doesn't hold. Of course, we can easily see that a counterexample would have to have at least one edge e. Consider the new graph $T - e$. This graph cannot be connected: if v, w are the vertices incident to e, then in T, since vew is a path from v to w, it is the unique one. But if $T - e$ were connected, a (v, w)-path in $T - e$ would give a (v, w)-path in T which didn't involve e, contradicting uniqueness.

We therefore have by Lemma 4.1.9 that $T - e$ is a disjoint union of two connected graphs $T - e = T_1 \cup T_2$. Since $T - e$ is acyclic, T_1, T_2 must be as well, and therefore they are both trees. By assumption of minimality of T, it follows that we have $e(T_1) + 1 = v(T_1)$ and $e(T_2) + 1 = v(T_2)$. But therefore we have

$$e(T) + 1 = \big(e(T_1) + e(T_2) + 1\big) + 1 = e(T_1) + 1 + e(T_2) + 1 = v(T_1) + v(T_2) = v(T)$$

as desired. □

5.2 Dijkstra's algorithm

Consider the following problem. Given a list of cities $c_1, c_2, \ldots, c_n$ and routes connecting certain cities, find the shortest route between two given cities c_i and c_j.

We can model this problem by letting the cities correspond to vertices of a connected graph G, and by letting the routes be the edges. To keep track of distances, we add the extra information of some "cost function:"

$$w : E_G \to \mathbb{R}_{\geq 0}$$

We may then ask the following question:

Question 5.2.1. *Given vertices $v, w \in V_G$, how do we find the shortest path between v an w.*

In practical sitations, particularly when the number of cities and paths is fairly large, it is not going to be effective to simply start listing paths in a brute force way. One should also consider how the information of the graph should be stored and accessed, for example on a computer. For example, if we represent the graph as a triple, as we have previously defined, this is not going to be particularly practical from the standpoint of implementation. For example, suppose we are starting at a vertex v and would like to start constructing our path. We would first have to start to look down our list of edges, and for every edge e, check and see whether or not v is incident to e, using the incidence function ψ. This is a great deal of work to do for every edge in the graph. Instead, it would be significantly more practical for this type of application to represent the graph as a list of tuples $L_v = (v, (e_1, v_1), (e_2, v_2), \ldots, (e_n, v_n))$, where the e_i are the edges incident to v, and where v, v_i are the vertices incident to e_i. In this way, starting from a vertex v, would mean starting with the list L_v, and being able to immediately access all incident edges, and look up the corresponding adjacent vertex in order to continue to build the path.

Let us not describe how to solve this problem. Starting from a vertex v, if we are searching for paths to eventually get to w, we will, in the process, successively build paths in various directions from v, finding minimal length paths to various other vertices in the graph, until we eventually reach our desired vertex w. In particular, the algorithm really will construct minimal length paths from a fixed starting vertex v to *all* other vertices of the graph. We are free, however, to stop the process once we have constructed a path to w.

How does this all relate to trees? Well, implicit in the process, we will be constructing *unique* paths from v to every other vertex, inductively. This will in fact result in building a spanning subgraph of G which will contain a unique path from v to every other vertex. It will follow quickly from Lemma 5.1.4, that the resulting spanning subgraph will in fact be a tree. In this way, one can see spanning trees as giving maps of efficient ways to traverse a graph, from a fixed starting point.

For a walk $W = v_1e_2v_2\cdots e_nv_n$ in G, we define the length of W to be

$$\ell(W) = \sum_{i=2}^{n} w(e_n).$$

For vertices $u, u' \in V_G$, let $d(u, u')$ be the distance between u and u'. That is,

$$d(u, u') = \min\{\ell(W) \mid W \text{ is a } (u, u')\text{-walk}\}$$

In this language our goal will be to construct, for a given v, w, a (v, w)-walk W such that $\ell(W) = d(v, w)$. Of course such a walk will necessarily be a path.

5.2.1 The algorithm

We will successsively build up a data set, which will consist of a subgraph $T \subset G$, together with the value $d(v, t)$ computed for each $t \in V_T$, and such that there will be a unique (v, t)-path W in T (which we will have, in the process of the algorithm, also computed), such that $\ell(W) = d(v, t)$.

For a given subgraph T, we will let

$$N(T)^\circ = \{u \in V_G \setminus V_T \mid u \text{ is adjacent to some vertex } u' \in V_T\}.$$

Remark 5.2.2. *For future reference, let us notice that the algorithm below constructs a sequence of subgraphs $T_0, T_1, \ldots$ each having $v(T_i) = e(T_i) + 1$.*

We proceed as follows:

1. begin with T_0 as the subgraph of G consisting only of the single vertex v. We have $d(v, v) = 0$ with the trivial path realizing this distance.

2. Having constructed T_i, for each $u \in N(T_i)^\circ$, let $v_1, \ldots, v_r \in V_{T_i}$ be the vertices of T_i which are adjacent to u. Choose an edge e incident to u, v_i, some i such that

$$d(v, v_i) + w(e)$$

   is minimal. Let $T_{i+1} = T + e$ Notice that if W is a shortest path from v to v_i in G which lies within T, then $W(v_ieu)$ is a minimal path from v to u.

3. Repeat the previous step until $w \in V_{T_i}$. Then set $T = T_i$.

Alternately, we may repeat the procedure until $V_G = V_{T_i}$. This will give a computation of a shortest path to every vertex of G from v.

Let's make sure that this algorithm does what it is supposed to do:

Lemma 5.2.3. *At a given stage, the new vertex u and edge e incident to u, v_i are such that, if W is a path in T_i from v to v_i of minimal length (i.e. such that $\ell(W) = d(v, v_i)$), then the new path Weu is a miminal length path from v to u.*

In fact, more is true: if u' is any vertex in $V_G \setminus V_{T_i}$, we have $\ell(Weu) = \ell(W) + w(e) \leq d(v, u')$. That is, u is at least as to v as any other vertex in G not already in T_i, and W in particular realizes a path of length at least as short as any of these distances.

Proof. We prove the second stronger statement. Suppose that W' is any walk from v to u' for u' any vertex in $V_G \setminus V_{T_i}$. We will show that $\ell(W') \leq \ell(W)$. Since W' starts at v and ends at u' not in T_i, there is some first vertex u'' which occurs along the walk W', and which is not in T_i if we let W'' be the first part of the walk W', going from v to u'' (omitting the part of the walk W' going from u'' to u'), we find that $\ell(W'') \leq \ell(W')$ (equality holding in the case that $u' = u''$). In particular, it suffices to show that $\ell(W) \leq \ell(W'')$. Since u'' is the first vertex of the walk not in T_i, the vertex just preceeding it must be in T_i, so W'' has the form $Ue'u''$ where U is a (v, w)-path with $w \in V_{T_i}$. By definition, it follows that $u'' \in N(T_i)^\circ$, and by the algorithm we know that by choice of u, for any $\ell(W)+w(e) = \ell(Weu) \leq \ell(U)+w(e') = \ell(W'')$ as claimed. ☐

Lemma 5.2.4. *The graph T produced by the algorithm above, is a spanning tree.*

Proof. By construction, it contains every vertex of G, and a path from v to each of its vertices. Therefore it is a connected spanning graph. To show it is a tree, we need to show that it is acyclic. If it did have a cycle, this would be added at some particular minimal stage, say going from T_i to T_{i+1}. That is to say, the cycle would necessarily involve the new edge e added. But by construction, the degree of the new added vertex u in T_{i+1} is 1 since the only edge it is incident to is e. But therefore it cannot be part of a cycle in T_{i+1}, giving a contradiction. ☐

Corollary 5.2.5. *Every connected graph has a spanning subtree $T < G$ with $v(T) = e(T)+1$.*

Chapter 6

Lecture 6: More Trees

6.1 Spanning trees and cycles

Proposition 6.1.1. *A connected graph G is a tree if and only if $e(G) + 1 = v(G)$.*

Proof. Suppose that G is connected with $e(G)+1 = v(G)$. Let $T < G$ be a spanning subtree with $e(T) + 1 = v(T)$. Since it is spanning, we have $v(T) = v(G) = e(G) + 1$. But therefore $e(G) = e(T)$ and so $T = G$, showing that G is a tree.

The converse is exactly the statement of Lemma 5.1.5. □

It follows that a tree can be characterized therefore as a minimal connected graph. As an illustration of this, one sees that a graph G is connected if and only if it contains a spanning subtree. On the one hand, such a spanning subtree would be a connected spanning subgraph, and it is clear that a graph with a spanning connected subgraph must itself be connected. On the other hand, if a graph is connected, then Dijkstra's algorithm provides us with a spanning subtree.

Lemma 6.1.2. *Let G be a simple graph, and T a spanning subgraph. Then for every $e \in E_G \setminus E_T$, $T + e$ contains a unique cycle.*

Proof. □

Chapter 7

Lecture 7: Connectivity, part 1

In this lecture, we will explore different notions of connectivity and their relations to each other. The main goal will be to discuss Menger's results, which relate two particular broad concepts: on the one hand, how easy or difficult it is to make a disconnect a graph by the deletion of edges or vertices, and on the other hand, how redundant the connections are within a graph, described in terms of the number of independent paths between pairs of vertices, where independence may be described either in terms of having no common edges or vertices, respectively[1]

7.1 Different notions of (dis-)connectivity

Before approaching Menger's results, however, we will start by simply relating the different notions of connectivity as measured by how one may make a graph disconnected by the removal of either edges or vertices. Recall (Definition 1.1.3), that a graph is trivial if it has only a single vertex.

Definition 7.1.1. *We say that the* ***connectivity*** *of a graph G, denoted $\kappa(G)$, is i, if i is the minimum number such that there exist vertices $v_1, \dots, v_i$ such that $G - \{v_i, \dots, v_i\}$ is either trivial or disconnected. If $\kappa(G) \leq k$, we say that G is* ***k-connected****.*

Definition 7.1.2. *We say that the* ***edge connectivity*** *of a graph G, denoted $\kappa'(G)$, is i, if i is the minimum number such that there exist edges $e_1, \dots, e_i$ such that $G - \{e_i, \dots, e_i\}$ is either trivial or disconnected. If $\kappa'(G) \leq k$, we say that G is* ***k-edge connected****.*

In particular, if G is itself disconnected, then $\kappa(G) = \kappa'(G) = 0$.

Intuitively, it should be easier to make a graph disconnected by the removal of vertices, as compared to the removal of edges, since the removal of a vertex also removes its incident

[1]idea for later: combine these notions: say a graph can be disconnected by removing some minimal collection of edges and vertices. how can we find an analogous notion of redundant paths (i.e. sharing no edges, and some maximal number of vertices...). raises the question of proportionality of vertices and edges.... also, is there a matroid interpretation for independent paths?

edges. In fact, this intuition is correct, as the following proposition shows. Recall that $\delta(G)$ is the minimum degree of a vertex of G.

Proposition 7.1.3. *Let G be a graph. Then $\delta(G) \geq \kappa'(G) \geq \kappa(G)$.*

Proof. The first inequality $\delta(G) \geq \kappa'(G)$ is straightforward: if G is trivial, then $\kappa'(G) = 0$ and the result is automatic. Otherwise, let $v \in G$ be a vertex of degree $\delta(G)$. Such a vertex is incident to at most $\delta(G)$ edges (exactly this many of none of the incident edges are loops). In this case, removal of these edges results in a graph in which v is now an isolated vertex, and hence a disconnected graph. It follows that $\delta(G) \geq \kappa'(G)$.

For the remaining inequality, we work by induction on $\kappa'(G)$, with base case $\kappa'(G) = 0$. Suppose that we know that the result is true for $(\ell - 1)$-connected graphs, and suppose that G is ℓ-connected. Let $E' \subset E(G)$ be a set of ℓ edges such that $G - E'$ is trivial or disconnected. Note that if $G - E'$ is trivial, then G itself must only have a single vertex and must therefore be trivial. In particular, we may assume that in fact $G - E'$ is disconnected. Also, by the minimality of the set E', it follows that E' contains no loops. Choose $e \in E'$ and let u, v be the two distinct incident vertices. Since we have $\kappa'(G-e) = \kappa'(G)-1 = \ell-1$, it follows by the induction hypothesis that $\kappa(G-e) \leq \ell-1$, and hence we may find vertices $V' = \{v_1, \ldots, v_{\ell-1}\} \subset V_G = V_{G-e}$ such that $(G-e) - V'$ is trivial or disconnected.

In the case that e is incident to one of the vertices of V', it follows that we actually have $(G-e)-V' = G-V'$, and hence V' would itself be a collection of vertices of G such that $G-V'$ is trivial or disconnected. In particular, in this case we would have $\kappa(G) \leq \ell-1 \leq \ell = \kappa'(G)$ as desired.

In the case that e is not incident ot a vertex of V', consider the graph $G - V'$. This graph becomes trivial or disconnected after the removal of a single edge e. Let w_1, w_2 be the vertices incident to e. If e is a loop then $G - V'$ was already trivial or disconnected and hence $\kappa(G) \leq \#V' = \ell - 1 \leq \kappa'(G)$. Let W_i be the connected component of w_i in $G - V' - e$ (note that these are the only connected components by Lemma 4.1.9). If both W_i are trivial with just the vertex w_i, then $G - V' - w_1$ is trivial and $\kappa(G) \leq \ell = \kappa(G')$. On the other hand, if $w_1 \neq u \in V_{W_1}$ then since u is not connected by any path to w_2 in $G - V' - e$ it cannot be connected by any path to w_2 in $G - V' - w_1$, since this is a subgraph of $G - V' - e$. Hence $G - V' - w_1$ is disconnected, and $\kappa(G) \leq \#(V \cup \{w_1\}) = \ell = \kappa'(G)$. $\square$

Chapter 8

Lecture 8: More Connectivity

8.1 Bonds, cotrees and cuts

Chapter 9

Tours

9.1 Regular graphs

Definition 9.1.1. *A graph G is called k-**regular** if every vertex $v \in V_G$ has degree k.*

Let's examine k regular simple graphs for small values of k. For example, a 0-regular graph is graph with no edges (just isolated vertices).

A connected 1-regular graph is easily seen to consist just of two vertices joined by a single edge. It follows that a general 1-regular graph, being a disjoint union of its components. It follows that for a graph G, a 1-regular spanning subgraph H corresponds to what is called a **matching**, namely, a way of writing the set of vertices V_G as a disjoint union $V_G = \bigcup P_i$ where each P_i is a pair of distinct adjacent vertices.

Lemma 9.1.2. *A connected 2-regular simple graph is isomorphic to a cycle graph C_n for some n.*

Definition 9.1.3. *A Hamiltonian cycle in a graph G is a cycle which passes through each vertex of G.*

We see that a connected 2-regular spanning subgraph of a graph G, corresponds to a Hamiltonian cycle.

Appendix A

Foundational notions

A.1 Sets and multisets

The substructure of the majority of modern mathematics is set theory. It therefore would behoove us to take a very slight digression into some useful concepts and notations.

Definition A.1.1. *A* ***set*** *is a collection of elements, is defined exactly by its elements. Two sets are equal if they contain the same elements.*

Notation A.1.2. *We will denote a set using "set notation." This consists of listing the elements of a set enclosed in braces, and separated by commas. Note that the order in which elements are written doesen't change the set. For example* $\{a, b, c\} = \{b, c, a\}$.

Definition A.1.3. *For a set* S*, its* ***power set*** $\mathscr{P}(S)$*, is the set whose elements are the subsets of* S

Definition A.1.4. *For a set* S*, we let* $\mathscr{P}_k(S)$ *denote its subsets with exactly* k *elements.*

Definition A.1.5. *For sets* S, T *we let* $S \times T$ *denote the set whose elements are ordered pairs* (s, t) *where* $s \in S$ *and* $t \in T$.

Definition A.1.6. *A* ***multiset*** $\mathscr{S}$ *is a pair* (S, m)*, where* S *is a set, and* m *is a function* $m : S \to \mathbb{Z}_{>0}$ *from* S *to the positive integers. For* $s \in S$*, we refer to* $m(s)$ *as the multiplicity of* s *in* $\mathscr{S}$*. We write* $s \in \mathscr{S}$*. We call* S *the underlying set of* $\mathscr{S}$.

For a multiset $\mathscr{S} = (S, m)$*, we will use the notation* $m_{\mathscr{S}}$ *to refer to* m.

Notation A.1.7. *We will write a multiset* S *by writing a list of its elements, with repetition, in a string, each elements arising in the string as many times as it its multiplicity. The order in which the elements are written doesen't matter. For example,* $abbbcc = abcbcb$.

Definition A.1.8. *Let* $\mathscr{S}$ *and* $\mathscr{T}$ *be multisets. We say that* $\mathscr{S} \subset \mathscr{T}$ *if the underlying set of* $\mathscr{S}$ *is contained in the underlying set of* $\mathscr{T}$.

If S is a set, we will also identify S with the multiset (S, m) defined by $m(s) = 1$ for each $s \in S$ (that is, S contains each of its elements exactly 1 time).

These mutisets are occasionally useful in combinatorics to think of the idea of sampling with replacement/repetition.

Definition A.1.9. *If $\mathscr{S} = (S, m)$ is a multiset, we define the* ***cardinality*** *of $\mathscr{S}$, denoted $\#\mathscr{S}$, to be*

$$\sum_{s \in S} m(s).$$

In particular, considering a set T as a multiset as described above, we have $\#T$ is exactly the number of elements of T.

Definition A.1.10. *Let S be a set. We write $\mathscr{R}(S)$ to denote the set of all mulisubsets of S, and $\mathscr{R}_k(S)$ the set of all multisubsets of S with cardinality exactly S.*

A.2 Relations

Definition A.2.1. *Recall that if X, Y are sets, a* ***relation*** *(from X to Y) is a subset $R \subset X \times Y$ of the product of X and Y. If an ordered pair $(x, y) \in R$ we say that x is related to y and write xRy.*

Notation A.2.2. *Frequently we will talk about a relation on a set X. This this is shorthand for a relation $R \subset X \times X$ from X to iteself.*

Two important types of relations are **functions** and **equivalence relations**, which we now describe.

A.2.1 Functions

Definition A.2.3. *A function from X to Y is a relation $f \subset X \times Y$ such that for every $x \in X$ there is exactly one $y \in Y$ such that xfy.*

Notation A.2.4. *If f is a function from x to y, we write $f(x)$ to denote the unique $y \in Y$ such that xfy.*

A.2.2 Equivalence Relations

Definition A.2.5. *Let R be a relation on X. We say that R is an* ***equivalence relation****, if the following properties hold*

1. *(reflexivity) for every $x \in X$, we have xRx,*
2. *(symmetry) for every $x, y \in X$, we have xRy if and only if yRx,*
3. *(transitivity) for every $x, y, z \in X$, whenever xRy and yRz we must also have xRz.*

Index

adjacent, 5
arrows, 8

cardinality, 28
circuit, 17
clique, 13
complement of a simple graph, 12
complete graph, 13
components, 16
connected, 16
connectivity, 23
cycle, 17
cycle graph, 13

degree, 9
digraph, *see* directed graph
directed graph, 8

edge chromatic number, 6
edge coloring, 6
edge connectivity, 23
equivalence relations, 28

forest, 18
functions, 28

graph, 4
    trivial, 5

incident, 5
indegree, 8
induced subgraph, 12
intersection (of graphs), 14
isomorphic graphs, 11
isomorphism, 11

leaf, 18
loop, 5

matching, 26
multiset, 27

origin (of a walk), 15
outdegree, 8

path, 17
planar graph, 5
power set, 27
proper, 6

regular, 26
relation, 28

set, 27
simple graph, 5
source (of an arrow), 8
spanning subgraph, 18
subgraph, 12

target (of an arrow), 8
terminus (of a walk), 15
trail, 17
tree, 18

union
    disjoint, 13
    edge disjoint, 13
union (of graphs), 13

vertex coloring, 6
    proper, 6

walk, 15
    closed, 17