

## GRAPH THEORY STUDY GUIDE

### 1. DEFINITIONS

**Definition 1** (Partition of A). A set  $\mathcal{A} = A_1, \dots, A_k$  of disjoint subsets of a set  $A$  is a partition of  $A$  if  $\cup \mathcal{A}$  of all the sets  $A_i \in \mathcal{A}$  and  $A_i \neq \emptyset$  for every  $i$ .

**Definition 2** (Vertex set). The set of vertices in a graph denoted by  $V(G)$ .

**Definition 3** (Edge set). The set of edges in a graph denoted by  $E(G)$ .

**Definition 4** (Order). the number of vertices of a graph  $G$  written  $|G|$ .

**Definition 5** (Incident). A vertex  $v$  is incident with an edge  $e$  if  $v \in e$ , then  $e$  is an edge at  $v$ .

**Definition 6** (Adjacent). Two vertices  $x, y$  of  $G$  are adjacent if  $xy$  is an edge of  $G$ .

**Definition 7** (Complete). If all vertices of  $G$  are pairwise adjacent, then  $G$  is complete.

**Definition 8** (Independent). A set of vertices or edges is independent if no two of its elements are adjacent.

**Definition 9** (Isomorphic). We call  $G$  and  $H$  isomorphic, and write  $G \simeq H$ , if there exists a bijection  $\phi : V \rightarrow V'$  with  $xy \in E \iff \phi(x)\phi(y) \in E'$  for all  $x, y$  in  $V$ .

**Definition 10** (Property). A class of graphs that is closed under isomorphism is called a graph property.

**Definition 11** (Invariant). A map taking graphs as arguments is called a graph invariant if it assigns equal values to isomorphic graphs.

**Definition 12** (Induced). If  $G \subseteq G'$  and  $G'$  contains all the edges  $xy \in E$  with  $x, y \in V'$ , then  $G'$  is an induced subgraph of  $G$ . We say that  $V'$  induces  $G'$  in  $G$ .

**Definition 13** (Spanning).  $G' \subseteq G$  is a spanning subgraph of  $G$  if  $V'$  spans all of  $G$  or  $V' = V$ .

**Definition 14** (Line Graph). The line graph  $L(G)$  of  $G$  is the graph of  $E$  in which  $x, y \in E$  are adjacent as vertices if and only if they are adjacent as edges in  $G$ .

**Definition 15** ( $N(G)$ ). the set of neighbors of a vertex  $v$ .

**Definition 16** (Degree). The degree ( $d(v)$ ) of a vertex  $v$  is the number  $|E(v)|$  of edges at  $v$  or the number of neighbors of  $v$ .

**Definition 17** (Isolated). A vertex of degree 0 is isolated.

**Definition 18** (Minimum degree).  $\delta(G)$  is the minimum degree of  $G$ .

**Definition 19** (Maximum degree).  $\Delta(G)$  = maximum degree of  $G$ .

**Definition 20** (Regular). all the vertices of  $G$  have the same degree  $k$ , then  $G$  is  $k$ -regular, or regular.

**Definition 21** (Average degree).  $d(G) = \frac{1}{|V|} \sum_{v \in V} d(v)$  and  $\delta(G) \leq d(G) \leq \Delta(G)$ .

**Definition 22** (Number of Edges).  $|E| = \frac{1}{2} \sum_{v \in V} d(v) = \frac{1}{2} d(G) * |V|$ .

**Definition 23** (Path). A path is a non-empty graph  $P = (V, E)$  of the form  $V = \{x_0, x_1, \dots, x_k\}$  and  $E = \{x_0x_1, \dots, x_{k-1}x_k\}$  where the  $x_i$  are all distinct.

**Definition 24** (H-path). Given a graph  $H$ , we call  $P$  an  $H$ -path if  $P$  is non-trivial and meets  $H$  exactly in its end points.

**Definition 25** (Cycle). If  $P = x_0, \dots, x_{k-1}$  is a path and  $k \geq 3$ , then the graph  $C = P + x_{k-1}x_0$  is called a cycle.

**Definition 26** (Girth). The minimum length of a cycle in a graph  $G$  is the girth ( $g(G)$ ).

**Definition 27** (Circumference). The maximum length of a cycle in  $G$  is its circumference.

**Definition 28** (Chord). An edge which joins two vertices of a cycle but is not itself an edge of the cycle is a chord of the cycle.

**Definition 29** (Induced cycle). an induced cycle in  $G$ , a cycle in  $G$  forming an induced subgraph , is one that has no chords.

**Definition 30** (Distance). The distance  $d_G(x, y)$  in  $G$  of two vertices  $x, y$  is the length of a shortest  $x - y$  path in  $G$ .

**Definition 31** (Diameter). The greatest distance between any two vertices in  $G$  is the diameter of  $G$ , denoted by  $\text{diam } G$ .

**Definition 32** (Central). a vertex is central in  $G$  if its greatest distance from any other vertex is as small as possible.

**Definition 33** (Radius). the greatest distance between the central vertex and from any other vertex denoted  $\text{rad } G$  where  $\text{rad } G = \min_{x \in V(G)} \max_{y \in V(G)} d_G(x, y)$  where  $\text{rad } G \leq \text{diam } G \leq 2\text{rad } G$ .

**Definition 34** (Walk). A walk of length  $k$  is a non-empty alternating sequence of vertices and edges in  $G$ .

**Definition 35** (Connected). A non-empty graph  $G$  is called connected if any two of its vertices are linked by a path in  $G$ .

**Definition 36** (Component). A maximal connected subgraph of  $G$  is called a component.

**Definition 37** (Separator). If  $A, B \subset V$  and  $X \subseteq V \cup E$  are such that every  $A - B$  path in  $G$  contains a vertex or an edge from  $X$ , we say that  $X$  separates the set  $A$  and  $B$  in  $G$ .

**Definition 38** (Cutvertex). A vertex which separates two other vertices of the same component is a cutvertex.

**Definition 39** (Bridge). An edge separating its ends is a bridge.

**Definition 40** ( $k$ -connected).  $G$  is called  $k$ -connected if  $|G| > k$  and  $G - X$  is connected for every set  $X \subseteq V$  with  $|X| < k$ . (No vertices of  $G$  are separated by fewer than  $k$  other vertices.)

**Definition 41** (Connectivity). The greatest integer  $k$  such that  $G$  is  $k$ -connected is the connectivity,  $\kappa(G)$ .

**Definition 42** ( $l$ -edge-connected). If  $|G| > 1$  and  $G - F$  is connected for every set  $F \subseteq E$  of fewer than  $l$  edges, then  $G$  is called  $l$ -edge-connected.

**Definition 43** (Edge-connectivity). The greatest integer  $l$  such that  $G$  is  $l$ -edge-connected is  $\lambda(G)$ .

**Definition 44** (Forest). An acyclic, graph, one not containing any cycles is called a forest.

**Definition 45** (Tree). A connected forest is called a tree.

**Definition 46** (Leaf). The vertices of degree 1 of a tree.

**Definition 47** (Tree-order). Writing  $x \geq y$  for  $x \in rTy$  then define a partial ordering on  $V(T)$ , the tree -order associated with  $T$  and  $r$ .

**Definition 48** (Normal Tree). A rooted tree  $T$  contained in a graph  $G$  is called normal in  $G$  if the ends of every  $T$ -path in  $G$  are comparable in the tree order of  $T$ .

**Definition 49** ( $r$ -partite). A graph  $G = (V, E)$  is called  $r$ -partitie if  $V$  admits a partition into  $r$  classes such that every edge has its ends in different classes.

**Definition 50** (Complete Partite). An  $r$ -partite graph in which every two vertices from different partitions classes are adjacent is called complete.

**Definition 51** (Contraction). Let  $e = xy$  be an edge of the graph  $G$ . By  $G/e$  we denote the graph obtained from  $G$  by contracting the edge  $e$  into a new vertex  $v_e$ .

**Definition 52** (Minor). If  $G = MX$  is a subgraph of another graph  $Y$ , we call  $X$  a minor of  $Y$ .

**Definition 53** (Eulerian). A closed walk in a graph is an Euler tour if it traverses every edge of the graph exactly once. It's called Eulerian if it admits an Euler tour.

**Definition 54** (Vertex Space). The vertex space of  $G$  is the vector space over the 2-element field of all functions  $V \rightarrow F_2$ ,  $\mathcal{V}(G)$ .

**Definition 55** (Edge Space). The function  $E \rightarrow F_2$  form the edge space of  $G$ ,  $\mathcal{E}(G)$ .

**Definition 56** (Cycle Space). The cycle space is the subspace of the edge space spanned by all the cycles in  $G$ .  $\mathcal{C} = \mathcal{C}(G)$ .

**Definition 57** (Cut Space). The cut edges of  $G$  form a subspace of  $\mathcal{E}(G)$  denoted by  $\mathcal{C}^*(G)$ .

**Definition 58** (Hypergraph). A hypergraph is a pair  $(V, E)$  of disjoint sets, where the elements of  $E$  are non-empty subset (of any cardinality) of  $V$ .

**Definition 59** (Directed). : A directed graph is a pair  $(V, E)$  of disjoint set (of vertices and edges) together with two maps  $\text{init} : E \rightarrow V$  and  $\text{ter} : E \rightarrow V$  assigning to every edge  $e$  an initial vertex  $\text{init}(e)$  and a terminal vertex  $\text{ter}(e)$ .

**Definition 60** (Orientation). A directed graph  $D$  is an orientation of an (undirected) graph  $G$  if  $V(D) = V(G)$  and  $E(D) = E(F)$  and if  $\{\text{init}(e), \text{ter}(e)\} = \{x, y\}$  for every edge  $e = xy$ .

**Definition 61** (Multigraph). A multigraph is a pair  $(V, E)$  of disjoint sets (of vertices and edges) together with a map  $E \rightarrow V \cup [V]^2$  assigning to every edge either one or two vertices , its ends.

**Definition 62** (Matching). A set  $M$  of independent edges in a graph  $G = (V, E)$  is called a matching.

**Definition 63** (Factor). A  $k$ -regular spanning subgraph is called  $k$ -factor.

**Definition 64** (Alternating Paths). A path in  $G$  which starts in  $A$  at an unmatched vertex and then contains, alternately, edges from  $E - M$  and from  $M$ , is an alternating path with respect to  $M$ .

**Definition 65** (Augmenting Path). An alternating path  $P$  that ends in an unmatched vertex of  $B$  is called an augmenting path.

**Definition 66** (Cover). Let us call a set  $U \subseteq V$  a (vertex) covering of  $E$  if every edge of  $G$  is incident with a vertex in  $U$ .

**Definition 67** ( $q(G)$ ). Given a graph  $G$ , let us denote by  $C_G$  the set of its components and by  $q(G)$  the number of its odd components, those of odd order.

**Definition 68** (Block). A maximal connected subgraph without a cutvertex is called a block.

**Definition 69** (Plane Graph). Graph drawing in the plane.

**Definition 70** (Planar). If  $G$  can be drawn in the plane without a crossing,  $G$  is planar.

**Definition 71** (Frontier). The frontier of a set  $X \subseteq \mathbb{R}^2$  is the set  $Y$  of all points  $y \in \mathbb{R}^2$  such that every neighborhood of  $y$  meets both  $X$  and  $\mathbb{R}^2 - X$ .

**Definition 72** (Geometric Graph). A Geometric Graph is a planar embedding of  $G$  with straight lines.

**Definition 73** (Topological Isomorphism). We call  $\sigma$  a topological isomorphism between  $H$  and  $H'$  plane graphs if there exists a homeomorphism  $\varphi : S^2 \rightarrow S^2$  such that  $\psi := \pi \circ \varphi \circ \pi^{-1}$  induces  $\sigma$  on  $V \cup E$ .

**Definition 74** (Combinatorial Isomorphism).  $\sigma$  can be extended to an incidence preserving map:  $\sigma : V \cup E \cup F \rightarrow V' \cup E' \cup F'$ . (preserves incidence of not only vertices and edges but also of vertices and edges with faces.)

**Definition 75** (Graph Theoretical Isomorphism).  $\sigma$  is a graph theoretical isomorphism of plane  $H$  and  $H'$  if  $\{\sigma(H[f]) : f \in F\} = \{H'[f'] : f \in F\}$ .

**Definition 76** (Simple). We call a subset  $\mathcal{F}$  of a graph's edge space  $\mathcal{E}(G)$  simple if every edge of  $G$  lies in at most two sets of  $\mathcal{F}$ .

**Definition 77** (Visible). Given a polygon  $P$ , vertices  $w, v, w$  is visible from  $v$ , if line segment  $\overline{vw} \cup \overline{P^c} = \emptyset$ .

**Definition 78** (Plane Multigraph). A plane multigraph is a pair  $G = (V, E)$  of finite sets (of vertices and edges, respectively) satisfying the following conditions:

- (1)  $V \subseteq \mathbb{R}^2$
- (2) every edge is either an arc between two vertices or a polygon containing exactly one vertex (its endpoints).
- (3) apart from its own endpoints(s), and edge contains no vertex and no point of any other edge.

**Definition 79** (Plane Dual). A plane dual  $(V^*, E^*) = G^*$  if there are bijections  $F \rightarrow V^*$ ,  $E \rightarrow E^*$ ,  $V \rightarrow F^*$ , where  $f \mapsto v^*(f)$ ,  $e \mapsto e^*$ ,  $v \mapsto f^*(v)$  satisfying the following conditions

- (1) i.  $v^*(v) \in f$  for all  $f \in F$
- (2)  $|e^* \cap G| = |\hat{e}^* \cap \hat{e}| = |e \cap G^*| = 1$  for all  $e \in E$ , and in each of  $e$  and  $e^*$  this point is an inner point of a straight line segment.
- (3)  $v \in f^*(v)$  for all  $v \in V$ .

**Definition 80** (Proper coloring). If  $c(v) \neq c(w)$  for all  $v$  adjacent  $w$ , then  $c$  is a proper coloring.

**Definition 81** ( $k$ -colorable).  $G$  is  $k$ -colorable if there exist proper coloring of the vertices with  $k$  colors.

**Definition 82** ( $k$ -edge-colorable).  $G$  is  $k$ -edge-colorable if there exists a proper edge coloring with  $k$  colors or proper coloring of line graph  $L(G)$ .

**Definition 83** ( $\chi(G)$ ). minimum  $k$  so that  $G$  is  $k$ -colorable or chromatic number of  $G$ .

**Definition 84** ( $\chi'(G)$ ). minimum  $k$  so that  $G$  is  $k$ -edge colorable or chromatic index of  $G$ .

**Definition 85** ( $c^{-1}(j)$ ). color class of all the vertices colored  $j$ .

**Definition 86** ( $k$ -constructible). For each  $k \in \mathbb{N}$ , let us define the class of  $k$ -constructible graphs recursively as follows:

- (1)  $K_k$  is  $k$ -constructible.
- (2) If  $G$  is  $k$ -constructible and  $x, y \in V(G)$  are non-adjacent, then also  $(G + xy)/xy$  is  $k$ -constructible.
- (3) If  $G_1, G_2$  are  $k$ -constructible and there are vertices  $x, y_1, y_2$  such that  $G_1 \cap G_2 = \{x\}$  and  $xy_1 \in E(G_1)$  and  $xy_2 \in E(G_2)$ , then also  $(G_1 \cup G_2) - xy_1 - xy_2 + y_1y_2$  is  $k$ -constructible.

## 2. THEOREMS

**Proposition 87.** The number of vertices of odd degree in a graph is always even.

**Proposition 88.** Every graph  $G$  with at least one edge has a subgraph  $H$  with  $\delta(H) > \epsilon(H) \geq \epsilon(G)$ .

**Proposition 89.** Every graph  $G$  contains a path of length  $\delta(G)$  and a cycle of length at least  $\delta(G)+1$  (provided that  $\delta(G) \geq 2$ ).

**Proposition 90.** Every graph  $G$  containing a cycle satisfies  $g(G) \leq 2\text{diam}(G) + 1$ .

**Proposition 91.** A graph  $G$  of radius at most  $k$  and maximum degree at most  $d \geq 3$  has fewer than  $\frac{d}{d-2}(d-1)^k$  vertices.

**Theorem 92.** Let  $G$  be a graph. If  $d(g) \geq d \geq 2$  and  $g(G) \geq g \in \mathbb{N}$  then  $|G| \geq n_0(d, g)$  where  $n_0(d, g) = 1 + d \sum_{i=0}^{r-1} (d-1)^i$  if  $g = 2r+1$  is odd;  $2 \sum_{i=0}^{r-1} (d-1)^i$  if  $g = 2r$  is even.

**Corollary 93.** If  $\delta(G) \geq 3$  then  $g(G) < 2\log|G|$ .

**Proposition 94.** The vertices of a connected graph  $G$  can always be enumerated say as  $v_1, \dots, v_n$  so that  $G_i = G[v_1, \dots, v_i]$  is connected for every  $i$ .

**Proposition 95.** If  $G$  is non-trivial then  $\kappa(G) \leq \lambda(G) \leq \delta(G)$ .

**Theorem 96.** Let  $0 \neq k \in \mathbb{N}$ . Every graph  $G$  with  $d(G) \geq 4k$  has  $(k+1)$ -connected subgraph  $H$  such that  $\epsilon(H) > \epsilon(G) - k$ .

**Theorem 97.** *The following assertions are equivalent for a graph  $T$ :*

- (1)  $T$  is a tree.
- (2) Any two vertices of  $T$  are linked by a unique path in  $T$ .
- (3)  $T$  is minimally connected, i.e.  $T$  is connected by  $T - e$  is disconnected for every edge  $e$  in  $T$ .
- (4)  $T$  is maximally acyclic, i.e.  $T$  contains no cycle but  $T + xy$  does, for any two non-adjacent vertices  $x, y \in T$ .

**Corollary 98.** *The vertices of a tree can always be enumerated, say as  $v_1, \dots, v_n$  so that every  $v_i$  with  $i \geq 2$  has a unique neighbor in  $\{v_1, \dots, v_{n-1}\}$ .*

**Corollary 99.** *A connected graph with  $n$  vertices is a tree if and only if it has  $n - 1$  edges.*

**Corollary 100.** *If  $T$  is a tree and  $G$  is any graph with  $\delta(G) \geq |T| - 1$ , then  $T \subseteq G$ , i.e.  $G$  has a subgraph isomorphic to  $T$ .*

**Lemma 101.** *Let  $T$  be a normal tree in  $G$ .*

- (1) Any two vertices  $x, y$  in  $T$  are separated in  $G$  by the set  $[x] \cap [y]$ .
- (2) If  $S \subseteq V(T) = V(G)$  and  $S$  is down-closed, then the components of  $G - S$  are spanned by the sets  $[x]$  with  $x$  minimal in  $T - S$ .

**Proposition 102.** *Every connected graph contains a normal spanning tree, with any specified vertex as its root.*

**Proposition 103.** *A graph is bipartite if and only if it contains no odd cycles.*

**Proposition 104.** (1) Every TX is also an MX; thus, every topological minor of a graph is also its (ordinary) minor.  
(2) If  $\Delta(X) \leq 3$ , then every MX contains a TX; thus every minor with maximum degree at most 3 of a graph is also its topological minor.

**Proposition 105.** *The minor relation  $\preccurlyeq$  and the topological minor relation are partial orderings on the class of finite graphs, ie. that are reflexive, antisymmetric and transitive.*

**Theorem 106.** *A connected graph is Eulerian if and only if Every vertex has even degree.*

**Proposition 107.** *The induced cycles in  $G$  generate its entire cycle space.*

**Proposition 108.** *The following assertions are equivalent for edge set  $F \subseteq E$*

- (1)  $F \in \mathcal{C}(G)$
- (2)  $F$  is a disjoint union of (edge sets of) cycles in  $G$ .
- (3) All vertex degrees of the graph  $(V, F)$  are even.

**Proposition 109.** *Together with  $\emptyset$ , the cuts in  $G$  form a subspace  $\mathcal{C}^*$  of  $\mathcal{E}$ . This space is generated by cuts of the form  $E(v)$ .*

**Lemma 110.** *Every cut is a disjoint union of bonds.*

**Theorem 111.** *The cycle space  $\mathcal{C}$  and the cut space  $\mathcal{C}^*$  of any graph satisfy  $\mathcal{C} = \mathcal{C}^{*\perp}$  and  $\mathcal{C}^* = \mathcal{C}^\perp$ .*

**Theorem 112.** *Let  $G$  be a connected graph and  $T \subseteq G$  a spanning tree. Then the corresponding fundamental cycles and cuts form a basis of  $\mathcal{C}(G)$  and of  $\mathcal{C}^*(G)$ , respectively. If  $G$  has  $n$  vertices and  $m$  edges, then  $\dim(\mathcal{C}(G)) = m - n + 1$  and  $\dim(\mathcal{C}^*(G)) = n - 1$ .*

**Theorem 113 (Konig).** *The maximum cardinality of a matching in  $G$  is equal to the minimum cardinality of a vertex cover of its edges.*

**Theorem 114 (Hall).**  *$G$  contains a matching of  $A$  if and only if  $|N(S)| \geq |S|$  for all  $S \subseteq A$ .*

**Corollary 115.** *If  $G$  is  $k$ -regular, with  $k \geq 1$ , then  $G$  has a 1-factor.*

**Corollary 116.** *Every regular graph of positive even degree has a 2-factor.*

**Theorem 117 (Tutte).** *A graph  $G$  has a 1-factor if and only if  $q(G - S) \leq |S|$  for  $S \subseteq V(G)$ .*

**Corollary 118 (Petersen).** *Every bridgeless cubic graph has a 1-factor.*

**Theorem 119.** Every graph  $G = (V, E)$  contains a vertex set  $S$  with the following two properties:

- (1)  $S$  is matchable to  $\mathcal{C}_{G-S}$ .
- (2) Every component of  $G - S$  is factor-critical.

Given any such set  $S$ , the graph  $G$  contains a 1-factor if and only if  $|S| = |\mathcal{C}_{G-S}|$

**Theorem 120.**  $G$  is 2-connected iff either i.  $G$  is a cycle or ii.  $G$  can be obtained from a 2-connected  $H$  by adding an  $H$ -path.

**Lemma 121.** If  $G$  is 3-connected and  $|G| > 4$ , then  $G$  has one edge s.t.  $G - e$  is 3-connected.

**Theorem 122.**  $G$  is 3-connected iff there exists a sequence of graphs  $G_0, \dots, G_n$  such that:

- (1)  $G_0 = K_4$ ;  $G_n = G$
- (2)  $G_{i+1}$  has an edge  $xy$  with  $d(x), d(y) \geq 3$  and  $G_i = G_{i+1}/xy$ ,  $\forall i < n$ .

**Theorem 123** (Tutte). The cycle space of a 3-connected graph is generated by non-separating induced cycle.

**Theorem 124** (Menger). Let  $G = (V, E)$  be a graph and  $A, B \subseteq V$ . Then the minimum number of vertices separating  $A$  from  $B$  in  $G$  is equal to the maximum number of disjoint  $A - B$  paths in  $G$ .

**Corollary 125.** Let  $a, b$  be distinct vertices of  $G$ .

- (1) If  $a, b \not\subseteq E$ , then minimum number of vertices  $\neq ab$  separating  $a$  from  $b$  equal maximum number of independent  $a - b$  path.
- (2) the minimum number of edges separating  $a$  from  $b$  in  $G$  is equal to the max number of edges disjoint  $a - b$  paths.

**Theorem 126** (Global Menger's Theorem). (1) A graph is  $k$ -connected if and only if it contains  $k$  independent path between any two vertices.

- (2) A graph is  $k$ -edge-connected if and only if it contains  $k$  edge-disjoint paths between any two vertices.

**Theorem 127** (Jordan Curve Theorem for Polygons). For every polygon  $P \subseteq \mathbb{R}^2$ , the set  $\mathbb{R}^2 - P$  has exactly two regions. Each of these has the entire polygon  $P$  as its frontier.

**Lemma 128.** Let  $P_1, P_2, P_3$  be three arcs, between the same two endpoints but otherwise disjoint:

- (1)  $\mathbb{R}^2 - (P_1 \cup P_2 \cup P_3)$  has exactly three regions, with frontiers  $P_1 \cup P_2$ ,  $P_2 \cup P_3$ , and  $P_1 \cup P_3$ .
- (2) If  $P$  is an arc between a point in  $P_1$  and a point in  $P_3$  whose interior lies in the region of  $\mathbb{R}^2 - (P_1 \cup P_3)$  that contains  $P_2$ , then  $P \cap P_2 \neq \emptyset$ .

**Lemma 129.** Let  $X_1, X_2 \subseteq \mathbb{R}^2$  be disjoint sets, each the union of finitely many points and arcs, and let  $P$  be an arc between a point in  $X_1$  and one in  $X_2$  whose interior  $P$  lies in a region  $O$  of  $\mathbb{R}^2 - (X_1 \cup X_2)$ . Then  $O \setminus P$  is a region of  $\mathbb{R}^2 - (X_1 \cup P \cup X_2)$ .

**Theorem 130.** Let  $\varphi : C_1 \rightarrow C_2$  be a homeomorphism between two circles on  $S^2$ , let  $O_1$  be a region of  $C_1$ , and let  $O_2$  be a region of  $C_2$ . Then  $\varphi$  can be extended to a homeomorphism  $C_1 \cup O_1 \rightarrow C_2 \cup O_2$ .

**Lemma 131.** Let  $G$  be a plane graph, and  $e$  an edge of  $G$ .

- (1) If  $X$  is the frontier of a face of  $G$ , then either  $e \subseteq X$  or  $X \cap e = \emptyset$ .
- (2) If  $e$  lies on a cycle  $C \subseteq G$ , then  $e$  lies on the frontier of exactly two faces of  $G$ , and these are contained in distinct faces of  $C$ .
- (3) If  $e$  lies on no cycle, then  $e$  lies on the frontier of exactly one face of  $G$ .

**Corollary 132.** The frontier of a face is always the point set of a subgraph.

**Proposition 133.** A plane forest has exactly one face.

**Lemma 134.** If a plane graph has different face with the same boundary, then the graph is a cycle.

**Proposition 135.** In a 2-connected plane graph, every face is bounded by a cycle.

**Proposition 136.** The face boundaries in a 3-connected plane graph are precisely its non-separating induced cycles.

**Proposition 137.** A plane graph of order at least 3 is maximally planar if and only if it is a plane triangulation.

**Theorem 138** (Euler's Formula). *Let  $G$  be a connected plane graph with  $n$  vertices,  $m$  edges, and  $l$  faces, then  $n - m + l = 2$ .*

**Corollary 139.** *A plane graph with  $n \geq 3$  vertices has at most  $3n - 6$  edges. Every plane triangulation with  $n$  vertices has  $3n - 6$ .*

**Corollary 140.** *A plane graph contains neither  $K_5$  nor  $K_{3,3}$  as a topological minor.*

**Theorem 141.** (1) *Every graph-theoretical isomorphism between two plane graphs is combinatorial. Its extension to a face bijection is unique if and only if the graph is not a cycle.*  
(2) *Every combinatorial isomorphism between two 2-connected plane graphs is topological.*

**Proposition 142.** *Every two embedding of a 3-connected Graph are equivalent.*

**Theorem 143** (Kuratowski). *The following are equivalent:*

- (1)  $G$  is planar.
- (2)  $G$  has no  $K_5$  and  $K_{3,3}$  minor.
- (3)  $G$  has no  $TK_5$  or  $TK_{3,3}$  minor.

**Lemma 144.** *Every 3-connected  $G$  with no  $K_5$  or  $K_{3,3}$  minor is planar.*

**Lemma 145.** *Let  $\chi$  be a set of 3-connected graphs. Let  $G$  be a graph with  $\kappa(G) \leq 2$  and  $G_1, G_2$  proper induced subgraph of  $G$  such that.  $G = G_1 \cup G_2$  and  $|G_1 \cap G_2| = \kappa(G)$ . If  $G$  is edge maximal with respect to not having a topological minor in  $\chi$ , then so are  $G_1$  and  $G_2$ , and  $G_1 \cup G_2 = K_2$ .*

**Lemma 146.** *If  $|G| \geq 4$  and  $G$  is edge-maximal with respect to having no  $TK_5$  or  $TK_{3,3}$  minor, then  $G$  is 3-connected.*

**Theorem 147** (Maclane).  *$G$  is planar if and only if its cycle space has a simple basis.*

**Theorem 148.** *A 3-connected graph is planar if and only if every edge lies on at most 2 non separating induced cycles.(equivalently: exactly).*

**Proposition 149.** *For any connected plane multigraph  $G$ , an edge set  $E \subseteq E(G)$  is the edge set of a cycle in  $G$  if and only if  $E^* := \{e^* | e \in E\}$  is a minimal cut in  $G^*$ .*

**Proposition 150.** *If  $G^*$  is an abstract dual of  $G$ , then the cut space of  $G^*$  is the cycle space of  $G$ ,  $\mathcal{C}^*(G^*) = \mathcal{C}(G)$ .*

**Theorem 151.** *A graph is planar if and only if it has an abstract dual.*

**Theorem 152.** *All planar graphs are 4-colorable.*

**Theorem 153.** *Every planar graph is 5-colourable.*

**Theorem 154.** *Every planar graph not containing a triangle is 3-colourable.*

**Proposition 155.** *Every graph  $G$  with  $m$  edges satisfies  $\chi(G) \leq \frac{1}{2} + \sqrt{(2m + \frac{1}{4})}$ .*

**Proposition 156.** *Every graph  $G$  satisfies  $\chi(G) \leq \text{col}(G) = \max \delta(H) | H \subseteq G + 1$ .*

**Corollary 157.** *Every graph  $G$  has a subgraph of minimum degree at least  $\chi(G) - 1$ .*

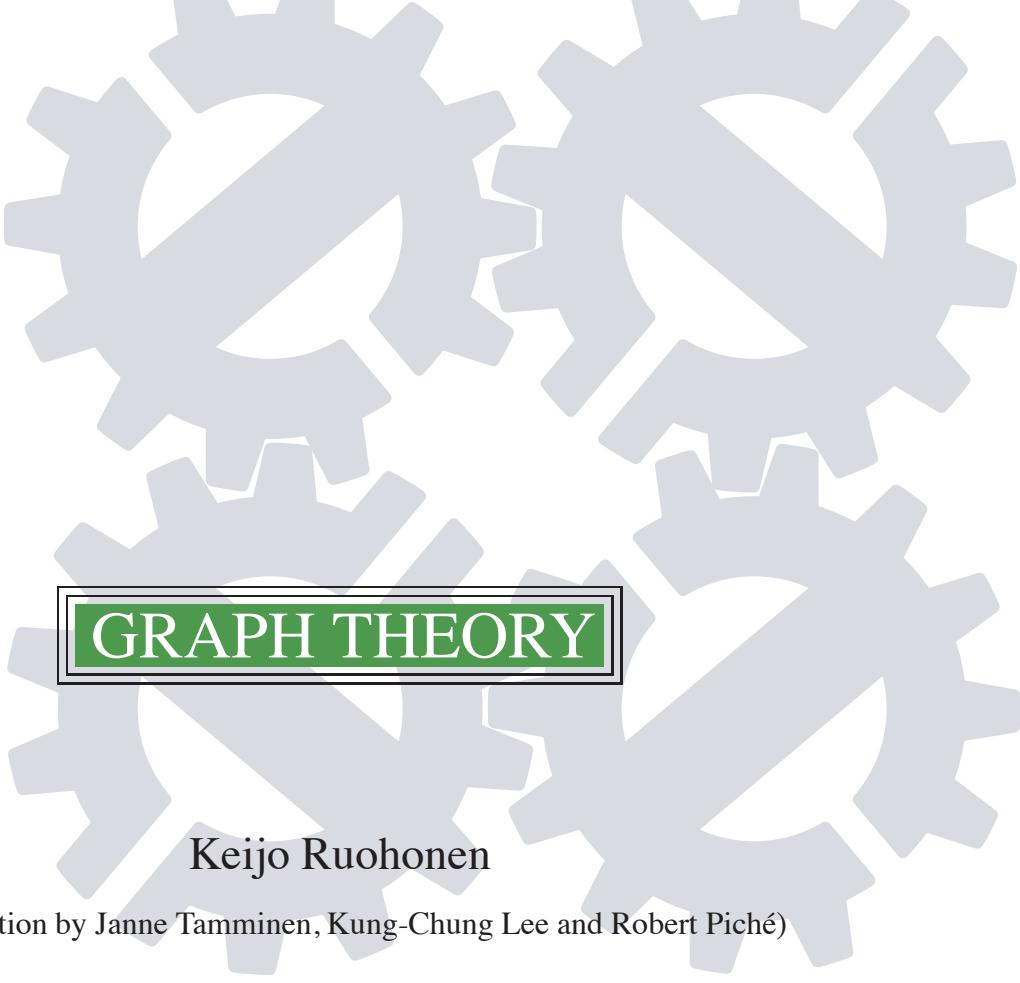
**Theorem 158** (Brooks). *Let  $G$  be a connected graph. If  $G$  is neither complete nor an odd cycle, then  $\chi(G) \leq \Delta(G)$ .*

**Theorem 159** (Erdős). *For every integer  $k$  there exists a graph  $G$  with girth  $g(G) > k$  and chromatic number  $\chi(G) > k$ .*

**Theorem 160** (Hajós). *Let  $G$  be a graph and  $k \in \mathbb{N}$ . Then  $\chi(G) \geq k$  if and only if  $G$  has a  $k$ -constructible subgraph.*

**Proposition 161** (Konig). *Every bipartite graph  $G$  satisfies  $\chi'(G) = \Delta(G)$ .*

**Theorem 162** (Vizing). *Every graph  $G$  satisfies  $\Delta(G) \leq \chi'(G) \leq \Delta(G) + 1$ .*



# GRAPH THEORY

Keijo Ruohonen

(Translation by Janne Tamminen, Kung-Chung Lee and Robert Piché)

2013

# Contents

<b>1</b>	<b>I DEFINITIONS AND FUNDAMENTAL CONCEPTS</b>
1	<b>1.1</b> Definitions
6	<b>1.2</b> Walks, Trails, Paths, Circuits, Connectivity, Components
10	<b>1.3</b> Graph Operations
14	<b>1.4</b> Cuts
18	<b>1.5</b> Labeled Graphs and Isomorphism
<b>20</b>	<b>II TREES</b>
20	<b>2.1</b> Trees and Forests
23	<b>2.2</b> (Fundamental) Circuits and (Fundamental) Cut Sets
<b>27</b>	<b>III DIRECTED GRAPHS</b>
27	<b>3.1</b> Definition
29	<b>3.2</b> Directed Trees
32	<b>3.3</b> Acyclic Directed Graphs
<b>34</b>	<b>IV MATRICES AND VECTOR SPACES OF GRAPHS</b>
34	<b>4.1</b> Matrix Representation of Graphs
36	<b>4.2</b> Cut Matrix
40	<b>4.3</b> Circuit Matrix
43	<b>4.4</b> An Application: Stationary Linear Networks
48	<b>4.5</b> Matrices over GF(2) and Vector Spaces of Graphs
<b>50</b>	<b>V GRAPH ALGORITHMS</b>
50	<b>5.1</b> Computational Complexity of Algorithms
52	<b>5.2</b> Reachability: Warshall's Algorithm
53	<b>5.3</b> Depth-First and Breadth-First Searches
61	<b>5.4</b> The Lightest Path: Dijkstra's Algorithm
63	<b>5.5</b> The Lightest Path: Floyd's Algorithm
66	<b>5.6</b> The Lightest Spanning Tree: Kruskal's and Prim's Algorithms
71	<b>5.7</b> The Lightest Hamiltonian Circuit (Travelling Salesman's Problem): The Annealing Algorithm and the Karp–Held Heuristics
76	<b>5.8</b> Maximum Matching in Bipartite Graphs: The Hungarian Algorithm
80	<b>5.9</b> Maximum Flow in a Transport Network: The Ford–Fulkerson Algorithm

85	<b>VI DRAWING GRAPHS</b>
85	<b>6.1</b> Planarity and Planar Embedding
90	<b>6.2</b> The Davidson–Harel Algorithm
92	<b>VII MATROIDS</b>
92	<b>7.1</b> Hereditary Systems
93	<b>7.2</b> The Circuit Matroid of a Graph
96	<b>7.3</b> Other Basic Matroids
98	<b>7.4</b> Greedy Algorithm
100	<b>7.5</b> The General Matroid
102	<b>7.6</b> Operations on Matroids
106	<b>References</b>
108	<b>Index</b>

## Foreword

These lecture notes were translated from the Finnish lecture notes for the TUT course on graph theory. The laborious bulk translation was taken care of by the students Janne Tamminen (TUT) and Kung-Chung Lee (visiting from the University of British Columbia). Most of the material was then checked by professor Robert Piché. I want to thank the translation team for their effort.

The notes form the base text for the course "MAT-62756 Graph Theory". They contain an introduction to basic concepts and results in graph theory, with a special emphasis put on the network-theoretic circuit-cut dualism. In many ways a model was the elegant and careful presentation of SWAMY & THULASIRAMAN, especially the older (and better) edition. There are of course many modern text-books with similar contents, e.g. the popular GROSS & YELLEN.

One of the usages of graph theory is to give a unified formalism for many very different-looking problems. It then suffices to present algorithms in this common formalism. This has lead to the birth of a special class of algorithms, the so-called *graph algorithms*. Half of the text of these notes deals with graph algorithms, again putting emphasis on network-theoretic methods. Only basic algorithms, applicable to problems of moderate size, are treated here. Special classes of algorithms, such as those dealing with sparse large graphs, "small-world" graphs, or parallel algorithms will not be treated. In these algorithms, data structure issues have a large role, too (see e.g. SKIENA).

The basis of graph theory is in combinatorics, and the role of "graphics" is only in visualizing things. Graph-theoretic applications and models usually involve connections to the "real world" on the one hand—often expressed in vivid graphical terms—and the definitional and computational methods given by the mathematical combinatoric and linear-algebraic machinery on the other. For many, this interplay is what makes graph theory so interesting. There is a part of graph theory which actually deals with graphical drawing and presentation of graphs, briefly touched in Chapter 6, where also simple algorithms are given for planarity testing and drawing. The presentation of the matter is quite superficial, a more profound treatment would require some rather deep results in topology and curve theory. Chapter 7 contains a brief introduction to matroids, a nice generalization and substitute for graphs in many ways.

Proofs of graph-theoretic results and methods are usually not given in a completely rigorous combinatoric form, but rather using the possibilities of visualization given by graphical presentations of graphs. This can lead to situations where the reader may not be completely convinced of the validity of proofs and derivations. One of the goals of a course in graph theory must then

be to provide the student with the correct "touch" to such seemingly loose methods of proof. This is indeed necessary, as a completely rigoristic mathematical presentation is often almost unreadable, whereas an excessively slack and lacunar presentation is of course useless.

Keijo Ruohonen

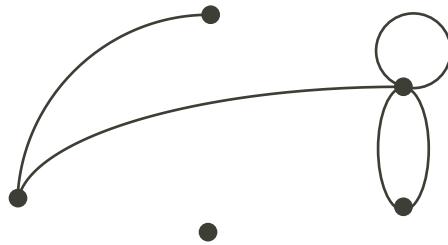
# Chapter 1

## Definitions and Fundamental Concepts

### 1.1 Definitions

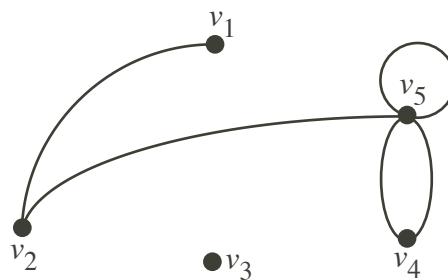
Conceptually, a *graph* is formed by *vertices* and *edges* connecting the vertices.

**Example.**



Formally, a graph is a pair of sets  $(V, E)$ , where  $V$  is the *set of vertices* and  $E$  is the *set of edges*, formed by pairs of vertices.  $E$  is a *multiset*, in other words, its elements can occur more than once so that every element has a *multiplicity*. Often, we label the vertices with letters (for example:  $a, b, c, \dots$  or  $v_1, v_2, \dots$ ) or numbers  $1, 2, \dots$ . Throughout this lecture material, we will label the elements of  $V$  in this way.

**Example.** (*Continuing from the previous example*) We label the vertices as follows:

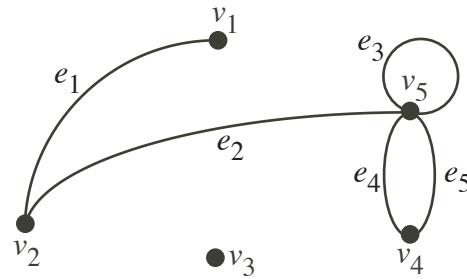


We have  $V = \{v_1, \dots, v_5\}$  for the vertices and  $E = \{(v_1, v_2), (v_2, v_5), (v_5, v_5), (v_5, v_4), (v_5, v_4)\}$  for the edges.

Similarly, we often label the edges with letters (for example:  $a, b, c, \dots$  or  $e_1, e_2, \dots$ ) or numbers  $1, 2, \dots$  for simplicity.

**Remark.** The two edges  $(u, v)$  and  $(v, u)$  are the same. In other words, the pair is not ordered.

**Example.** (Continuing from the previous example) We label the edges as follows:



So  $E = \{e_1, \dots, e_5\}$ .

We have the following terminologies:

1. The two vertices  $u$  and  $v$  are *end vertices* of the edge  $(u, v)$ .
2. Edges that have the same end vertices are *parallel*.
3. An edge of the form  $(v, v)$  is a *loop*.
4. A graph is *simple* if it has no parallel edges or loops.
5. A graph with no edges (i.e.  $E$  is empty) is *empty*.
6. A graph with no vertices (i.e.  $V$  and  $E$  are empty) is a *null graph*.
7. A graph with only one vertex is *trivial*.
8. Edges are *adjacent* if they share a common end vertex.
9. Two vertices  $u$  and  $v$  are *adjacent* if they are connected by an edge, in other words,  $(u, v)$  is an edge.
10. The *degree* of the vertex  $v$ , written as  $d(v)$ , is the number of edges with  $v$  as an end vertex. By convention, we count a loop twice and parallel edges contribute separately.
11. A *pendant vertex* is a vertex whose degree is 1.
12. An edge that has a pendant vertex as an end vertex is a *pendant edge*.
13. An *isolated vertex* is a vertex whose degree is 0.

**Example.** (Continuing from the previous example)

- $v_4$  and  $v_5$  are end vertices of  $e_5$ .
- $e_4$  and  $e_5$  are parallel.
- $e_3$  is a loop.
- The graph is not simple.
- $e_1$  and  $e_2$  are adjacent.

- $v_1$  and  $v_2$  are adjacent.
- The degree of  $v_1$  is 1 so it is a pendant vertex.
- $e_1$  is a pendant edge.
- The degree of  $v_5$  is 5.
- The degree of  $v_4$  is 2.
- The degree of  $v_3$  is 0 so it is an isolated vertex.

In the future, we will label graphs with letters, for example:

$$G = (V, E).$$

The *minimum degree* of the vertices in a graph  $G$  is denoted  $\delta(G)$  ( $= 0$  if there is an isolated vertex in  $G$ ). Similarly, we write  $\Delta(G)$  as the *maximum degree* of vertices in  $G$ .

**Example.** (Continuing from the previous example)  $\delta(G) = 0$  and  $\Delta(G) = 5$ .

**Remark.** In this course, we only consider finite graphs, i.e.  $V$  and  $E$  are finite sets.

Since every edge has two end vertices, we get

**Theorem 1.1.** The graph  $G = (V, E)$ , where  $V = \{v_1, \dots, v_n\}$  and  $E = \{e_1, \dots, e_m\}$ , satisfies

$$\sum_{i=1}^n d(v_i) = 2m.$$

**Corollary.** Every graph has an even number of vertices of odd degree.

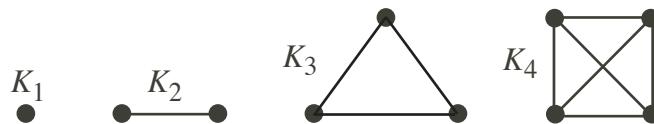
*Proof.* If the vertices  $v_1, \dots, v_k$  have odd degrees and the vertices  $v_{k+1}, \dots, v_n$  have even degrees, then (Theorem 1.1)

$$d(v_1) + \dots + d(v_k) = 2m - d(v_{k+1}) - \dots - d(v_n)$$

is even. Therefore,  $k$  is even. □

**Example.** (Continuing from the previous example) Now the sum of the degrees is  $1 + 2 + 0 + 2 + 5 = 10 = 2 \cdot 5$ . There are two vertices of odd degree, namely  $v_1$  and  $v_5$ .

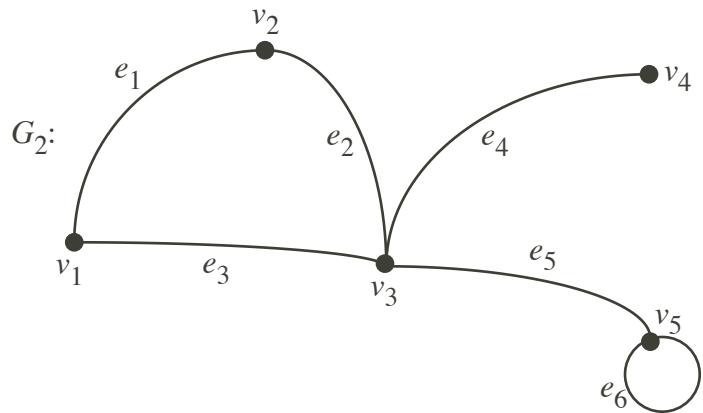
A simple graph that contains every possible edge between all the vertices is called a *complete graph*. A complete graph with  $n$  vertices is denoted as  $K_n$ . The first four complete graphs are given as examples:



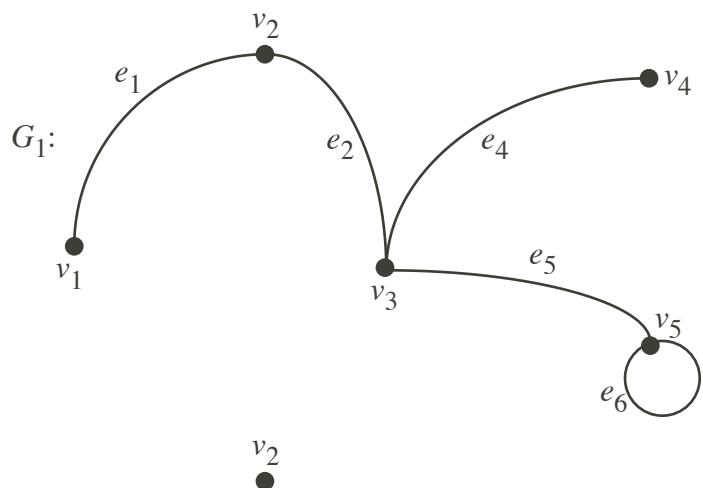
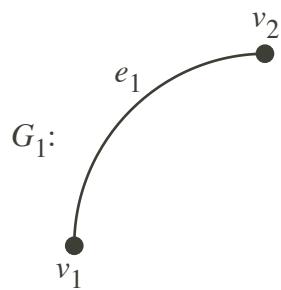
The graph  $G_1 = (V_1, E_1)$  is a *subgraph* of  $G_2 = (V_2, E_2)$  if

1.  $V_1 \subseteq V_2$  and
2. Every edge of  $G_1$  is also an edge of  $G_2$ .

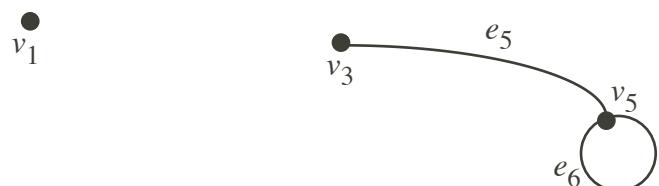
**Example.** We have the graph



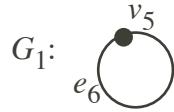
and some of its subgraphs are



$G_1$ :



and

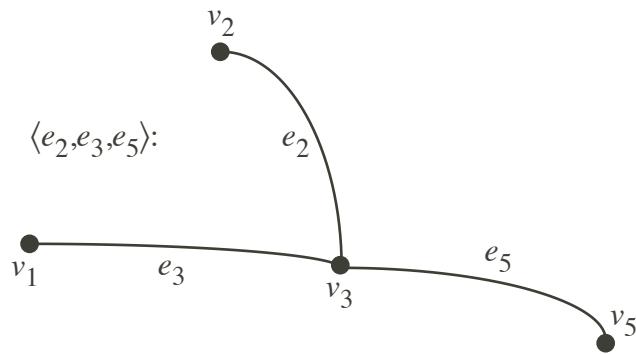


The *subgraph* of  $G = (V, E)$  induced by the edge set  $E_1 \subseteq E$  is:

$$G_1 = (V_1, E_1) =_{\text{def.}} \langle E_1 \rangle,$$

where  $V_1$  consists of every end vertex of the edges in  $E_1$ .

**Example.** (Continuing from above) From the original graph  $G$ , the edges  $e_2$ ,  $e_3$  and  $e_5$  induce the subgraph

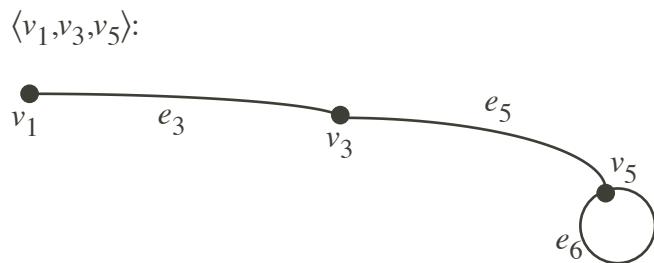


The *subgraph* of  $G = (V, E)$  induced by the vertex set  $V_1 \subseteq V$  is:

$$G_1 = (V_1, E_1) =_{\text{def.}} \langle V_1 \rangle,$$

where  $E_1$  consists of every edge between the vertices in  $V_1$ .

**Example.** (Continuing from the previous example) From the original graph  $G$ , the vertices  $v_1$ ,  $v_3$  and  $v_5$  induce the subgraph



A complete subgraph of  $G$  is called a *clique* of  $G$ .

## 1.2 Walks, Trails, Paths, Circuits, Connectivity, Components

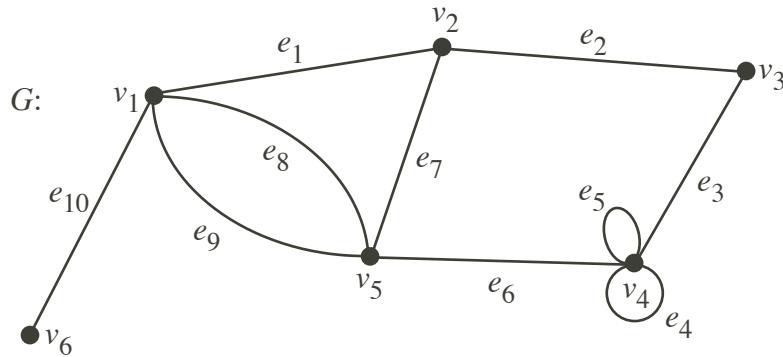
**Remark.** There are many different variations of the following terminologies. We will adhere to the definitions given here.

A *walk* in the graph  $G = (V, E)$  is a finite sequence of the form

$$v_{i_0}, e_{j_1}, v_{i_1}, e_{j_2}, \dots, e_{j_k}, v_{i_k},$$

which consists of alternating vertices and edges of  $G$ . The walk starts at a vertex. Vertices  $v_{i_{t-1}}$  and  $v_{i_t}$  are end vertices of  $e_{j_t}$  ( $t = 1, \dots, k$ ).  $v_{i_0}$  is the *initial vertex* and  $v_{i_k}$  is the *terminal vertex*.  $k$  is the *length* of the walk. A zero length walk is just a single vertex  $v_{i_0}$ . It is allowed to visit a vertex or go through an edge more than once. A walk is *open* if  $v_{i_0} \neq v_{i_k}$ . Otherwise it is *closed*.

**Example.** In the graph



the walk

$$v_2, e_7, v_5, e_8, v_1, e_8, v_5, e_6, v_4, e_5, v_4, e_5, v_4$$

is open. On the other hand, the walk

$$v_4, e_5, v_4, e_3, v_3, e_2, v_2, e_7, v_5, e_6, v_6, v_4$$

is closed.

A walk is a *trail* if any edge is traversed at most once. Then, the number of times that the vertex pair  $u, v$  can appear as consecutive vertices in a trail is at most the number of parallel edges connecting  $u$  and  $v$ .

**Example.** (Continuing from the previous example) The walk in the graph

$$v_1, e_8, v_5, e_9, v_1, e_1, v_2, e_7, v_5, e_6, v_4, e_5, v_4, e_4, v_4$$

is a trail.

A trail is a *path* if any vertex is visited at most once except possibly the initial and terminal vertices when they are the same. A closed path is a *circuit*. For simplicity, we will assume in the future that a circuit is not empty, i.e. its length  $\geq 1$ . We identify the paths and circuits with the subgraphs induced by their edges.

**Example.** (Continuing from the previous example) The walk

$$v_2, e_7, v_5, e_6, v_4, e_3, v_3$$

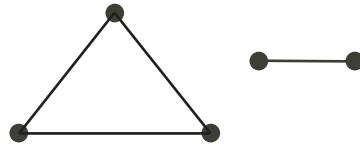
is a path and the walk

$$v_2, e_7, v_5, e_6, v_4, e_3, v_3, e_2, v_2$$

is a circuit.

The walk starting at  $u$  and ending at  $v$  is called an  $u-v$  walk.  $u$  and  $v$  are connected if there is a  $u-v$  walk in the graph (then there is also a  $u-v$  path!). If  $u$  and  $v$  are connected and  $v$  and  $w$  are connected, then  $u$  and  $w$  are also connected, i.e. if there is a  $u-v$  walk and a  $v-w$  walk, then there is also a  $u-w$  walk. A graph is connected if all the vertices are connected to each other. (A trivial graph is connected by convention.)

**Example.** The graph



is not connected.

The subgraph  $G_1$  (not a null graph) of the graph  $G$  is a component of  $G$  if

1.  $G_1$  is connected and
2. Either  $G_1$  is trivial (one single isolated vertex of  $G$ ) or  $G_1$  is not trivial and  $G_1$  is the subgraph induced by those edges of  $G$  that have one end vertex in  $G_1$ .

Different components of the same graph do not have any common vertices because of the following theorem.

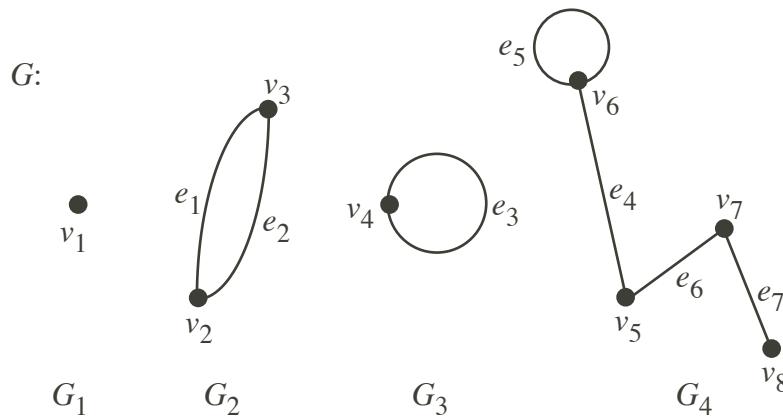
**Theorem 1.2.** If the graph  $G$  has a vertex  $v$  that is connected to a vertex of the component  $G_1$  of  $G$ , then  $v$  is also a vertex of  $G_1$ .

*Proof.* If  $v$  is connected to vertex  $v'$  of  $G_1$ , then there is a walk in  $G$

$$v = v_{i_0}, e_{j_1}, v_{i_1}, \dots, v_{i_{k-1}}, e_{j_k}, v_{i_k} = v'.$$

Since  $v'$  is a vertex of  $G_1$ , then (condition #2 above)  $e_{j_k}$  is an edge of  $G_1$  and  $v_{i_{k-1}}$  is a vertex of  $G_1$ . We continue this process and see that  $v$  is a vertex of  $G_1$ .  $\square$

**Example.**



The components of  $G$  are  $G_1, G_2, G_3$  and  $G_4$ .

**Theorem 1.3.** Every vertex of  $G$  belongs to exactly one component of  $G$ . Similarly, every edge of  $G$  belongs to exactly one component of  $G$ .

*Proof.* We choose a vertex  $v$  in  $G$ . We do the following as many times as possible starting with  $V_1 = \{v\}$ :

- (\*) If  $v'$  is a vertex of  $G$  such that  $v' \notin V_1$  and  $v'$  is connected to some vertex of  $V_1$ , then  $V_1 \leftarrow V_1 \cup \{v'\}$ .

Since there is a finite number of vertices in  $G$ , the process stops eventually. The last  $V_1$  induces a subgraph  $G_1$  of  $G$  that is the component of  $G$  containing  $v$ .  $G_1$  is connected because its vertices are connected to  $v$  so they are also connected to each other. Condition #2 holds because we can not repeat (\*). By Theorem 1.2,  $v$  does not belong to any other component.

The edges of the graph are incident to the end vertices of the components.  $\square$

Theorem 1.3 divides a graph into distinct components. The proof of the theorem gives an algorithm to do that. We have to repeat what we did in the proof as long as we have free vertices that do not belong to any component. Every isolated vertex forms its own component. A connected graph has only one component, namely, itself.

A graph  $G$  with  $n$  vertices,  $m$  edges and  $k$  components has the *rank*

$$\rho(G) = n - k.$$

The *nullity* of the graph is

$$\mu(G) = m - n + k.$$

We see that  $\rho(G) \geq 0$  and  $\rho(G) + \mu(G) = m$ . In addition,  $\mu(G) \geq 0$  because

**Theorem 1.4.**  $\rho(G) \leq m$

*Proof.* We will use the second principle of induction (strong induction) for  $m$ .

Induction Basis:  $m = 0$ . The components are trivial and  $n = k$ .

Induction Hypothesis: The theorem is true for  $m < p$ . ( $p \geq 1$ )

Induction Statement: The theorem is true for  $m = p$ .

Induction Statement Proof: We choose a component  $G_1$  of  $G$  which has at least one edge. We label that edge  $e$  and the end vertices  $u$  and  $v$ . We also label  $G_2$  as the subgraph of  $G$  and  $G_1$ , obtained by removing the edge  $e$  from  $G_1$  (but not the vertices  $u$  and  $v$ ). We label  $G'$  as the graph obtained by removing the edge  $e$  from  $G$  (but not the vertices  $u$  and  $v$ ) and let  $k'$  be the number of components of  $G'$ . We have two cases:

1.  $G_2$  is connected. Then,  $k' = k$ . We use the Induction Hypothesis on  $G'$ :

$$n - k = n - k' = \rho(G') \leq m - 1 < m.$$

2.  $G_2$  is not connected. Then there is only one path between  $u$  and  $v$ :

$$u, e, v$$

and no other path. Thus, there are two components in  $G_2$  and  $k' = k + 1$ . We use the Induction Hypothesis on  $G'$ :

$$\rho(G') = n - k' = n - k - 1 \leq m - 1.$$

Hence  $n - k \leq m$ . □

These kind of combinatorial results have many consequences. For example:

**Theorem 1.5.** *If  $G$  is a connected graph and  $k \geq 2$  is the maximum path length, then any two paths in  $G$  with length  $k$  share at least one common vertex.*

*Proof.* We only consider the case where the paths are not circuits (Other cases can be proven in a similar way.). Consider two paths of  $G$  with length  $k$ :

$$v_{i_0}, e_{j_1}, v_{i_1}, e_{j_2}, \dots, e_{j_k}, v_{i_k} \quad (\text{path } p_1)$$

and

$$v_{i'_0}, e_{j'_1}, v_{i'_1}, e_{j'_2}, \dots, e_{j'_k}, v_{i'_k} \quad (\text{path } p_2).$$

Let us consider the counter hypothesis: The paths  $p_1$  and  $p_2$  do not share a common vertex. Since  $G$  is connected, there exists an  $v_{i_0}-v_{i'_k}$  path. We then find the last vertex on this path which is also on  $p_1$  (at least  $v_{i_0}$  is on  $p_1$ ) and we label that vertex  $v_{i_t}$ . We find the first vertex of the  $v_{i_t}-v_{i'_k}$  path which is also on  $p_2$  (at least  $v_{i'_k}$  is on  $p_2$ ) and we label that vertex  $v_{i_s}$ . So we get a  $v_{i_t}-v_{i'_s}$  path

$$v_{i_t}, e_{j''_1}, \dots, e_{j''_\ell}, v_{i'_s}.$$

The situation is as follows:

$$\begin{aligned} & v_{i_0}, e_{j_1}, v_{i_1}, \dots, v_{i_t}, e_{j_{t+1}}, \dots, e_{j_k}, v_{i_k} \\ & \quad e_{j''_1} \\ & \quad \vdots \\ & \quad e_{j''_\ell} \\ & v_{i'_0}, e_{j'_1}, v_{i'_1}, \dots, v_{i'_s}, e_{j'_{s+1}}, \dots, e_{j'_k}, v_{i'_k} \end{aligned}$$

From here we get two paths:  $v_{i_0}-v_{i'_k}$  path and  $v_{i'_0}-v_{i_k}$  path. The two cases are:

- $t \geq s$ : Now the length of the  $v_{i_0}-v_{i'_k}$  path is  $\geq k + 1$ . ✓<sup>1</sup>
- $t < s$ : Now the length of the  $v_{i'_0}-v_{i_k}$  path is  $\geq k + 1$ . ✓

□

A graph is *circuitless* if it does not have any circuit in it.

**Theorem 1.6.** *A graph is circuitless exactly when there are no loops and there is at most one path between any two given vertices.*

*Proof.* First let us assume  $G$  is circuitless. Then, there are no loops in  $G$ . Let us assume the counter hypothesis: There are two different paths between distinct vertices  $u$  and  $v$  in  $G$ :

$$u = v_{i_0}, e_{j_1}, v_{i_1}, e_{j_2}, \dots, e_{j_k}, v_{i_k} = v \quad (\text{path } p_1)$$

and

$$u = v_{i'_0}, e_{j'_1}, v_{i'_1}, e_{j'_2}, \dots, e_{j'_\ell}, v_{i'_\ell} = v \quad (\text{path } p_2)$$

(here we have  $i_0 = i'_0$  and  $i_k = i'_\ell$ ), where  $k \geq \ell$ . We choose the smallest index  $t$  such that

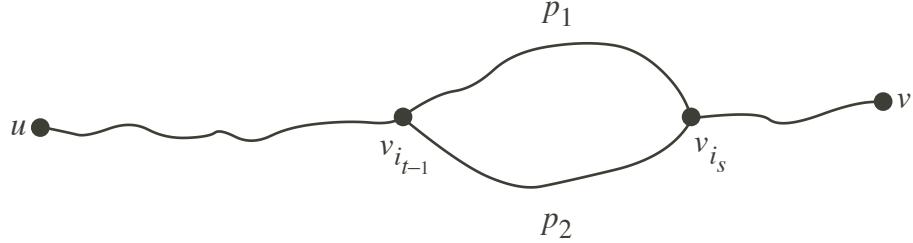
$$v_{i_t} \neq v_{i'_t}.$$

There is such a  $t$  because otherwise

---

<sup>1</sup>From now on, the symbol ✓ means contradiction. If we get a contradiction by proceeding from the assumptions, the hypothesis must be wrong.

1.  $k > \ell$  and  $v_{i_k} = v = v_{i'_\ell} = v_{i_\ell}$  ( $\checkmark$ ) or
2.  $k = \ell$  and  $v_{i_0} = v_{i'_0}, \dots, v_{i_\ell} = v_{i'_\ell}$ . Then, there would be two parallel edges between two consecutive vertices in the path. That would imply the existence of a circuit between two vertices in  $G$ .  $\checkmark$



We choose the smallest index  $s$  such that  $s \geq t$  and  $v_{i_s}$  is in the path  $p_2$  (at least  $v_{i_k}$  is in  $p_2$ ). We choose an index  $r$  such that  $r \geq t$  and  $v_{i_r} = v_{i_s}$  (it exists because  $p_1$  is a path). Then,

$$v_{i_{t-1}}, e_{j_t}, \dots, e_{j_s}, v_{i_s} (= v_{i_r}), e_{j'_r}, \dots, e_{j'_t}, v_{i'_{t-1}} (= v_{i_{t-1}})$$

is a circuit.  $\checkmark$  (Verify the case  $t = s = r$ .)

Let us prove the reverse implication. If the graph does not have any loops and no two distinct vertices have two different paths between them, then there is no circuit. For example, if

$$v_{i_0}, e_{j_1}, v_{i_1}, e_{j_2}, \dots, e_{j_k}, v_{i_k} = v_{i_0}$$

is a circuit, then either  $k = 1$  and  $e_{j_1}$  is a loop ( $\checkmark$ ), or  $k \geq 2$  and the two vertices  $v_{i_0}$  and  $v_{i_1}$  are connected by two distinct paths

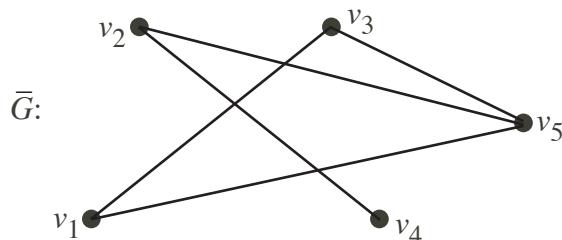
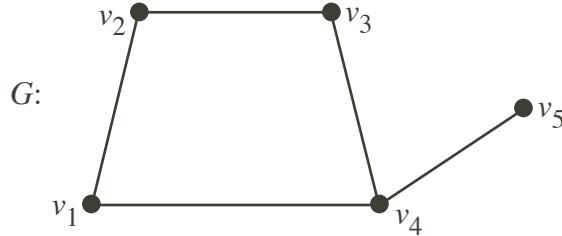
$$v_{i_0}, e_{j_1}, v_{i_1} \quad \text{and} \quad v_{i_1}, e_{j_2}, \dots, e_{j_k}, v_{i_k} = v_{i_0} \quad (\checkmark).$$

□

### 1.3 Graph Operations

The *complement* of the simple graph  $G = (V, E)$  is the simple graph  $\bar{G} = (V, \bar{E})$ , where the edges in  $\bar{E}$  are exactly the edges not in  $G$ .

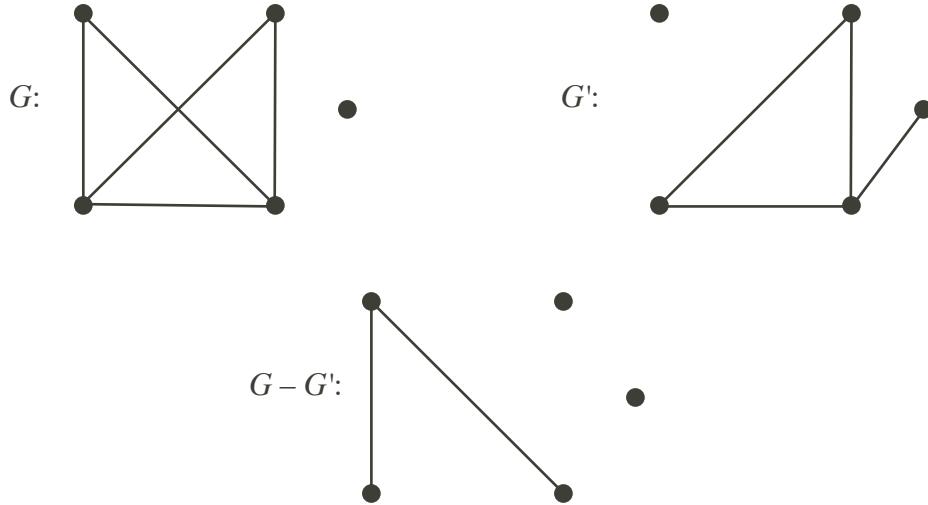
**Example.**



**Example.** The complement of the complete graph  $K_n$  is the empty graph with  $n$  vertices.

Obviously,  $\overline{\overline{G}} = G$ . If the graphs  $G = (V, E)$  and  $G' = (V', E')$  are simple and  $V' \subseteq V$ , then the *difference* graph is  $G - G' = (V, E'')$ , where  $E''$  contains those edges from  $G$  that are not in  $G'$  (simple graph).

**Example.**



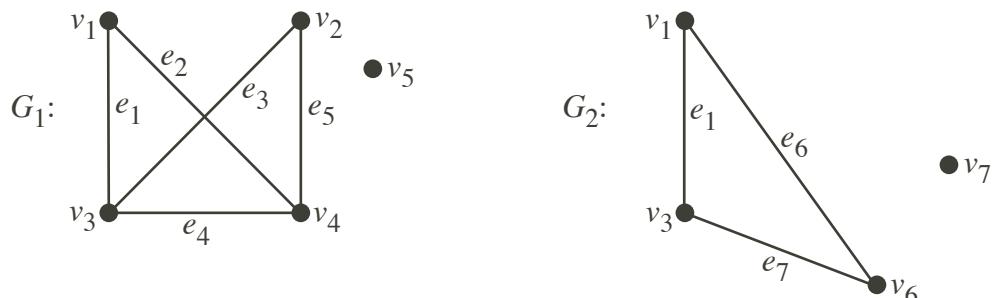
Here are some binary operations between two simple graphs  $G_1 = (V_1, E_1)$  and  $G_2 = (V_2, E_2)$ :

- The *union* is  $G_1 \cup G_2 = (V_1 \cup V_2, E_1 \cup E_2)$  (simple graph).
- The *intersection* is  $G_1 \cap G_2 = (V_1 \cap V_2, E_1 \cap E_2)$  (simple graph).
- The *ring sum*  $G_1 \oplus G_2$  is the subgraph of  $G_1 \cup G_2$  induced by the edge set  $E_1 \oplus E_2$  (simple graph). Note! The set operation  $\oplus$  is the *symmetric difference*, i.e.

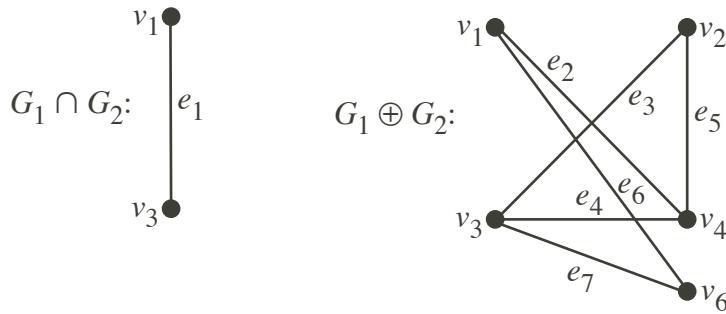
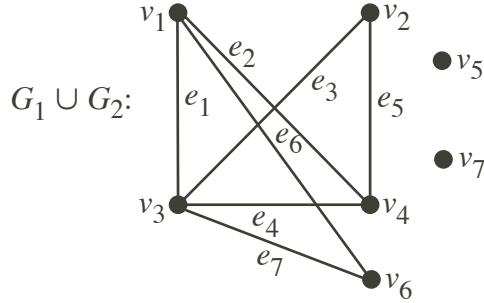
$$E_1 \oplus E_2 = (E_1 - E_2) \cup (E_2 - E_1).$$

Since the ring sum is a subgraph induced by an edge set, there are no isolated vertices. All three operations are commutative and associative.

**Example.** For the graphs



we have



**Remark.** The operations  $\cup$ ,  $\cap$  and  $\oplus$  can also be defined for more general graphs other than simple graphs. Naturally, we have to "keep track" of the multiplicity of the edges:

$\cup$  : The multiplicity of an edge in  $G_1 \cup G_2$  is the larger of its multiplicities in  $G_1$  and  $G_2$ .

$\cap$  : The multiplicity of an edge in  $G_1 \cap G_2$  is the smaller of its multiplicities in  $G_1$  and  $G_2$ .

$\oplus$  : The multiplicity of an edge in  $G_1 \oplus G_2$  is  $|m_1 - m_2|$ , where  $m_1$  is its multiplicity in  $G_1$  and  $m_2$  is its multiplicity in  $G_2$ .

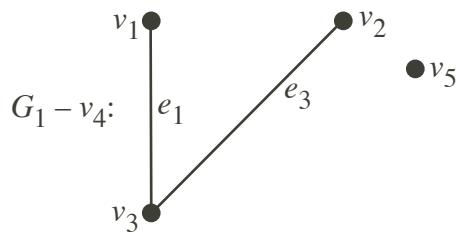
(We assume zero multiplicity for the absence of an edge.) In addition, we can generalize the difference operation for all kinds of graphs if we take account of the multiplicity. The multiplicity of the edge  $e$  in the difference  $G - G'$  is

$$m_1 - m_2 = \begin{cases} m_1 - m_2, & \text{if } m_1 \geq m_2 \\ 0, & \text{if } m_1 < m_2 \end{cases} \quad (\text{also known as the proper difference}),$$

where  $m_1$  and  $m_2$  are the multiplicities of  $e$  in  $G_1$  and  $G_2$ , respectively.

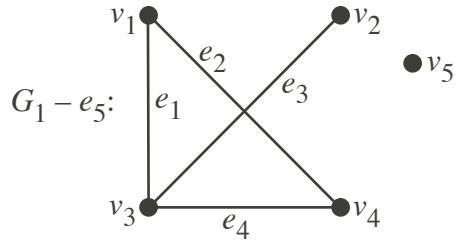
If  $v$  is a vertex of the graph  $G = (V, E)$ , then  $G - v$  is the subgraph of  $G$  induced by the vertex set  $V - \{v\}$ . We call this operation the *removal of a vertex*.

**Example.** (Continuing from the previous example)



Similarly, if  $e$  is an edge of the graph  $G = (V, E)$ , then  $G - e$  is graph  $(V, E')$ , where  $E'$  is obtained by removing  $e$  from  $E$ . This operation is known as *removal of an edge*. We remark that we are not talking about removing an edge as in Set Theory, because the edge can have nonunit multiplicity and we only remove the edge once.

**Example.** (*Continuing from the previous example*)



If  $u$  and  $v$  are two distinct vertices of the graph  $G = (V, E)$ , then we can *short-circuit* the two vertices  $u$  and  $v$  and obtain the graph  $(V', E')$ , where

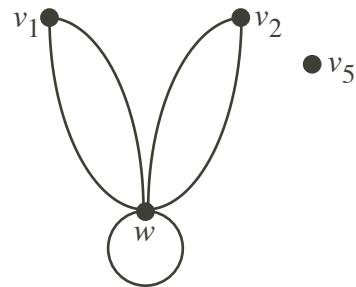
$$V' = (V - \{u, v\}) \cup \{w\} \quad (w \notin V \text{ is the "new" vertex})$$

and

$$\begin{aligned} E' = & (E - \{(v', u), (v', v) \mid v' \in V\}) \cup \{(v', w) \mid (v', u) \in E \text{ or } (v', v) \in E\} \\ & \cup \{(w, w) \mid (u, u) \in E \text{ or } (v, v) \in E\} \end{aligned}$$

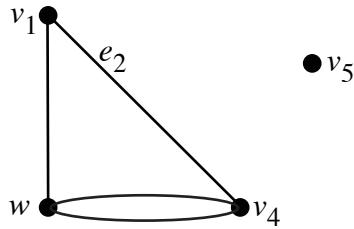
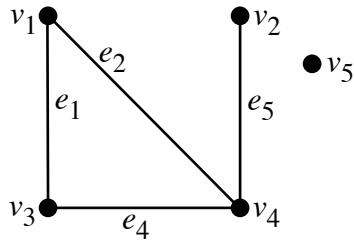
(Recall that the pair of vertices corresponding to an edge is not ordered). *Note!* We have to maintain the multiplicity of the edges. In particular, the edge  $(u, v)$  becomes a loop.

**Example.** (*Continuing from the previous example*) Short-circuit  $v_3$  and  $v_4$  in the graph  $G_1$ :



In the graph  $G = (V, E)$ , *contracting* the edge  $e = (u, v)$  (not a loop) means the operation in which we first remove  $e$  and then short-circuit  $u$  and  $v$ . (Contracting a loop simply removes that loop.)

**Example.** (*Continuing from the previous example*) We contract the edge  $e_3$  in  $G_1$  by first removing  $e_3$  and then short-circuiting  $v_2$  and  $v_3$ .

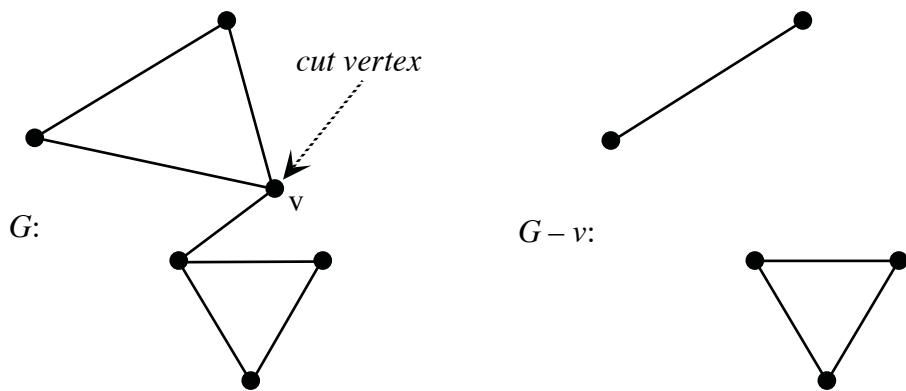


**Remark.** If we restrict short-circuiting and contracting to simple graphs, then we remove loops and all but one of the parallel edges between end vertices from the results.

## 1.4 Cuts

A vertex  $v$  of a graph  $G$  is a *cut vertex* or an *articulation vertex* of  $G$  if the graph  $G - v$  consists of a greater number of components than  $G$ .

**Example.**  $v$  is a cut vertex of the graph below:

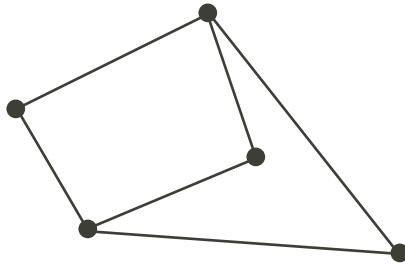


(Note! Generally, the only vertex of a trivial graph is not a cut vertex, neither is an isolated vertex.)

A graph is *separable* if it is not connected or if there exists at least one cut vertex in the graph. Otherwise, the graph is *nonseparable*.

**Example.** The graph  $G$  in the previous example is separable.

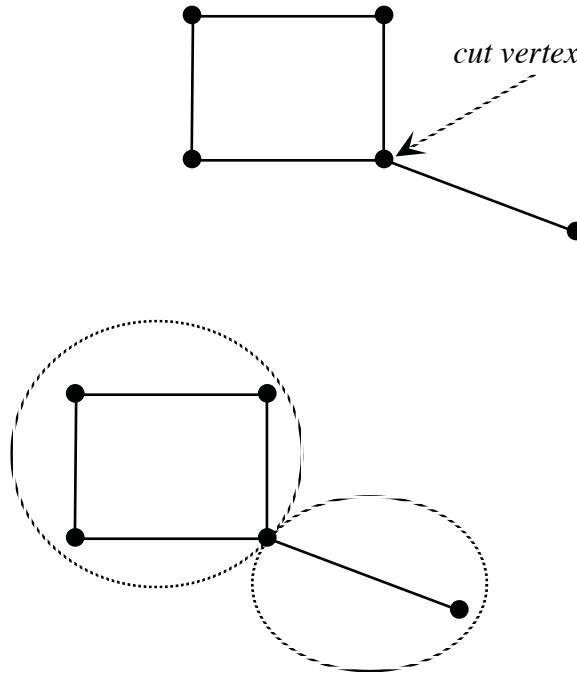
**Example.** The graph below is nonseparable.



A *block* of the graph  $G$  is a subgraph  $G_1$  of  $G$  (not a null graph) such that

- $G_1$  is nonseparable, and
- if  $G_2$  is any other subgraph of  $G$ , then  $G_1 \cup G_2 = G_1$  or  $G_1 \cup G_2$  is separable (think about that!).

**Example.** The graph below is separable:



**Theorem 1.7.** The vertex  $v$  is a cut vertex of the connected graph  $G$  if and only if there exist two vertices  $u$  and  $w$  in the graph  $G$  such that

- (i)  $v \neq u, v \neq w$  and  $u \neq w$ , but
- (ii)  $v$  is on every  $u-w$  path.

*Proof.* First, let us consider the case that  $v$  is a cut-vertex of  $G$ . Then,  $G - v$  is not connected and there are at least two components  $G_1 = (V_1, E_1)$  and  $G_2 = (V_2, E_2)$ . We choose  $u \in V_1$  and  $w \in V_2$ . The  $u-w$  path is in  $G$  because it is connected. If  $v$  is not on this path, then the path is also in  $G - v$  ( $\checkmark$ ). The same reasoning can be used for all the  $u-w$  paths in  $G$ .

If  $v$  is in every  $u-w$  path, then the vertices  $u$  and  $w$  are not connected in  $G - v$ .  $\square$

**Theorem 1.8.** *A nontrivial simple graph has at least two vertices which are not cut vertices.*

*Proof.* We will use induction for the graph  $G$  with  $n$  vertices.

Induction Basis: The case  $n = 2$  is obviously true.

Induction Hypothesis: The theorem is true for  $n \leq k$ . ( $k \geq 2$ )

Induction Statement: The theorem is true for  $n = k + 1$ .

Induction Statement Proof: If there are no cut vertices in  $G$ , then it is obvious. Otherwise, we consider a cut vertex  $v$  of  $G$ . Let  $G_1, \dots, G_m$  be the components of  $G - v$  (so  $m \geq 2$ ). Every component  $G_i$  falls into one of the two cases:

1.  $G_i$  is trivial so the only vertex of  $G_i$  is a pendant vertex or an isolated vertex of  $G$  but it is not a cut vertex of  $G$ .
2.  $G_i$  is not trivial. The Induction Hypothesis tells us that there exist two vertices  $u$  and  $w$  in  $G_i$  which are not cut vertices of  $G_i$ . If  $v$  and  $u$  (respectively  $v$  and  $w$ ) are not adjacent in  $G$ , then  $u$  (respectively  $w$ ) is not a cut vertex in  $G$ . If both  $v$  and  $u$  as well as  $u$  and  $w$  are adjacent in  $G$ , then  $u$  and  $w$  can not be cut vertices of  $G$ .  $\square$

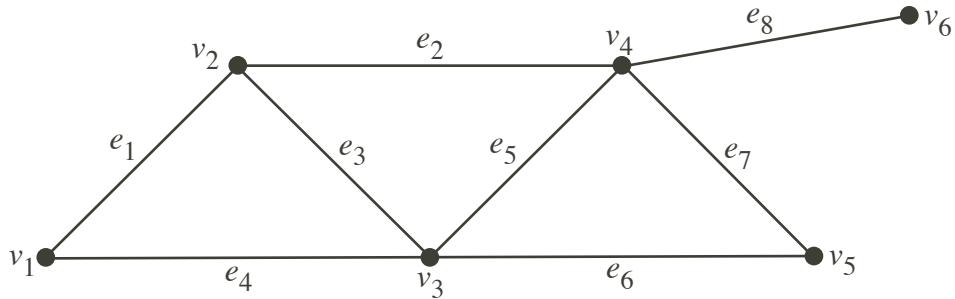
A *cut set* of the connected graph  $G = (V, E)$  is an edge set  $F \subseteq E$  such that

1.  $G - F$  (remove the edges of  $F$  one by one) is not connected, and
2.  $G - H$  is connected whenever  $H \subset F$ .

**Theorem 1.9.** *If  $F$  is a cut set of the connected graph  $G$ , then  $G - F$  has two components.*

*Proof.* Let  $F = \{e_1, \dots, e_k\}$ . The graph  $G - \{e_1, \dots, e_{k-1}\}$  is connected (and so is  $G$  if  $k = 1$ ) by condition #2. When we remove the edges from the connected graph, we get at most two components.  $\square$

**Example.** In the graph



$\{e_1, e_4\}$ ,  $\{e_6, e_7\}$ ,  $\{e_1, e_2, e_3\}$ ,  $\{e_8\}$ ,  $\{e_3, e_4, e_5, e_6\}$ ,  $\{e_2, e_5, e_7\}$ ,  $\{e_2, e_5, e_6\}$  and  $\{e_2, e_3, e_4\}$  are cut sets. Are there other cut sets?

In a graph  $G = (V, E)$ , a pair of subsets  $V_1$  and  $V_2$  of  $V$  satisfying

$$V = V_1 \cup V_2, \quad V_1 \cap V_2 = \emptyset, \quad V_1 \neq \emptyset, \quad V_2 \neq \emptyset,$$

is called a *cut* (or a *partition*) of  $G$ , denoted  $\langle V_1, V_2 \rangle$ . Usually, the cuts  $\langle V_1, V_2 \rangle$  and  $\langle V_2, V_1 \rangle$  are considered to be the same.

**Example.** (Continuing from the previous example)  $\langle \{v_1, v_2, v_3\}, \{v_4, v_5, v_6, v_7\} \rangle$  is a cut.

We can also think of a cut as an edge set:

$$\text{cut } \langle V_1, V_2 \rangle = \{\text{those edges with one end vertex in } V_1 \text{ and the other end vertex in } V_2\}.$$

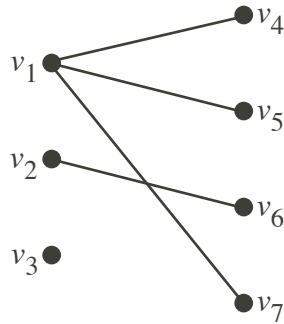
(Note! This edge set does not define  $V_1$  and  $V_2$  uniquely so we can not use this for the definition of a cut.)

Using the previous definitions and concepts, we can easily prove the following:

1. The cut  $\langle V_1, V_2 \rangle$  of a connected graph  $G$  (considered as an edge set) is a cut set if and only if the subgraphs induced by  $V_1$  and  $V_2$  are connected, i.e.  $G - \langle V_1, V_2 \rangle$  has two components.
2. If  $F$  is a cut set of the connected graph  $G$  and  $V_1$  and  $V_2$  are the vertex sets of the two components of  $G - F$ , then  $\langle V_1, V_2 \rangle$  is a cut and  $F = \langle V_1, V_2 \rangle$ .
3. If  $v$  is a vertex of a connected (nontrivial) graph  $G = (V, E)$ , then  $\langle \{v\}, V - \{v\} \rangle$  is a cut of  $G$ . It follows that the cut is a cut set if the subgraph (i.e.  $G - v$ ) induced by  $V - \{v\}$  is connected, i.e. if  $v$  is *not* a cut vertex.

If there exists a cut  $\langle V_1, V_2 \rangle$  for the graph  $G = (V, E)$  so that  $E = \langle V_1, V_2 \rangle$ , i.e. the cut (considered as an edge set) includes every edge, then the graph  $G$  is *bipartite*.

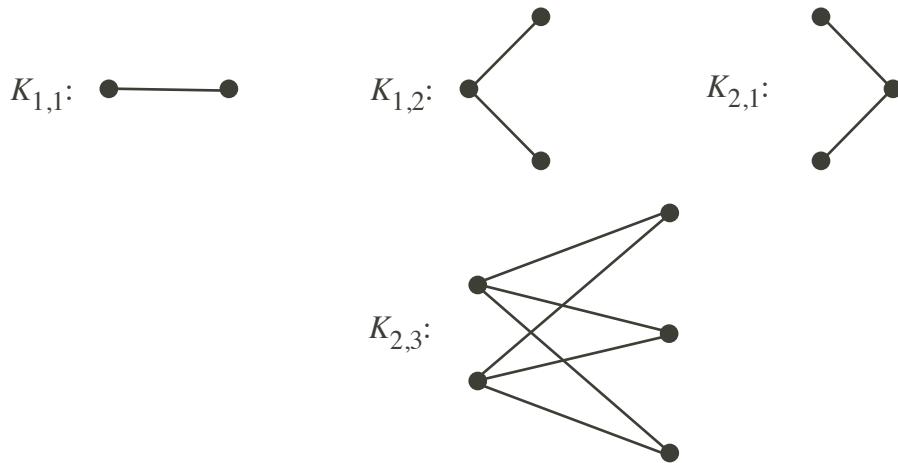
**Example.** The graph



is bipartite.  $V_1 = \{v_1, v_2, v_3\}$  and  $V_2 = \{v_4, v_5, v_6, v_7\}$ .

A simple bipartite graph is called a *complete bipartite graph* if we can not possibly add any more edges to the edge set  $\langle V_1, V_2 \rangle$ , i.e. the graph contains exactly all edges that have one end vertex in  $V_1$  and the other end vertex in  $V_2$ . If there are  $n$  vertices in  $V_1$  and  $m$  vertices in  $V_2$ , we denote it as  $K_{n,m}$  (cf. complete graph).

**Example.**



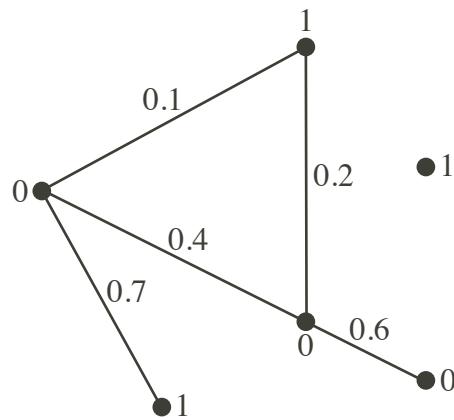
(Usually  $K_{n,m}$  and  $K_{m,n}$  are considered to be the same.)

## 1.5 Labeled Graphs and Isomorphism

By a *labeling of the vertices* of the graph  $G = (V, E)$ , we mean a mapping  $\alpha : V \rightarrow A$ , where  $A$  is called the *label set*. Similarly, a *labeling of the edges* is a mapping  $\beta : E \rightarrow B$ , where  $B$  is the label set. Often, these labels are numbers. Then, we call them *weights* of vertices and edges. In a weighted graph, the weight of a path is the sum of the weights of the edges traversed.

The labeling of the vertices (respectively edges) is *injective* if distinct vertices (respectively edges) have distinct labels. An injective labeling is *bijective* if there are as many labels in  $A$  (respectively in  $B$ ) as the number of vertices (respectively edges).

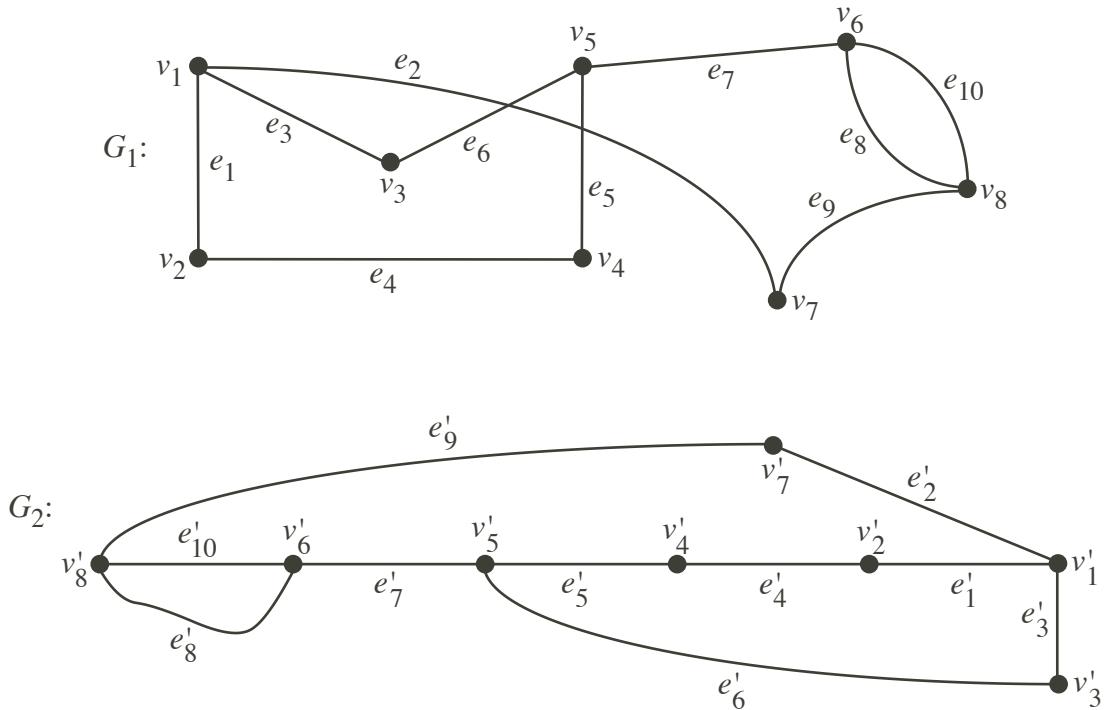
**Example.** If  $A = \{0, 1\}$  and  $B = \mathbb{R}$ , then in the graph,



the labeling of the edges (weights) is injective but not the labeling of the vertices.

The two graphs  $G_1 = (V_1, E_1)$  and  $G_2 = (V_2, E_2)$  are *isomorphic* if labeling the vertices of  $G_1$  bijectively with the elements of  $V_2$  gives  $G_2$ . (Note! We have to maintain the multiplicity of the edges.)

**Example.** The graphs  $G_1$  and  $G_2$  are isomorphic and the vertex labeling  $v_i \mapsto v'_i$  and edge labeling  $e_j \mapsto e'_j$  define the isomorphism.



Determining whether or not two graphs are isomorphic is a well researched<sup>2</sup> problem. It differs significantly from other problems in graph theory and network analysis. In addition, it has a lot to do with group theory in algebra. The problem is important in the theory of Computational Complexity. For example, refer to KÖBLER, J. & SCHÖNING, U. & TORÁN, J.: *The Graph Isomorphism Problem. Its Structural Complexity*. Birkhäuser (1993).

---

<sup>2</sup>Maybe too well, cf. READ, R.C. & CORNEIL, D.G.: The Graph Isomorphism Disease. *Journal of Graph Theory* **1** (1977), 339–363.

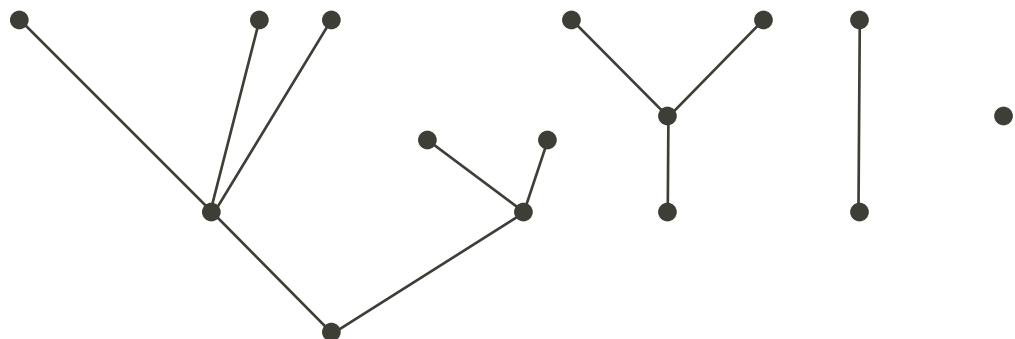
# Chapter 2

## Trees

### 2.1 Trees and Forests

A *forest* is a circuitless graph. A *tree* is a connected forest. A *subforest* is a subgraph of a forest. A connected subgraph of a tree is a *subtree*. Generally speaking, a subforest (respectively subtree) of a graph is its subgraph, which is also a forest (respectively tree).

**Example.** Four trees which together form a forest:

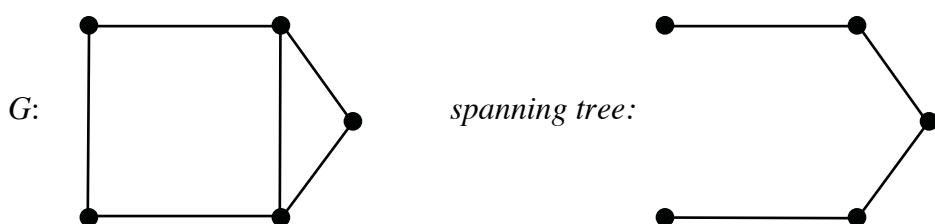


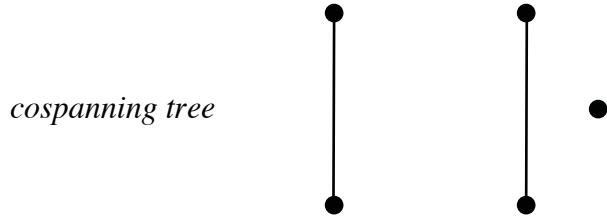
A *spanning tree* of a connected graph is a subtree that includes all the vertices of that graph. If  $T$  is a spanning tree of the graph  $G$ , then

$$G - T =_{\text{def.}} T^*$$

is the *cospanning tree*.

**Example.**





The edges of a spanning tree are called *branches* and the edges of the corresponding cospanning tree are called *links* or *chords*.

**Theorem 2.1.** *If the graph  $G$  has  $n$  vertices and  $m$  edges, then the following statements are equivalent:*

- (i)  $G$  is a tree.
- (ii) There is exactly one path between any two vertices in  $G$  and  $G$  has no loops.
- (iii)  $G$  is connected and  $m = n - 1$ .
- (iv)  $G$  is circuitless and  $m = n - 1$ .
- (v)  $G$  is circuitless and if we add any new edge to  $G$ , then we will get one and only one circuit.

*Proof.* (i) $\Rightarrow$ (ii): If  $G$  is a tree, then it is connected and circuitless. Thus, there are no loops in  $G$ . There exists a path between any two vertices of  $G$ . By Theorem 1.6, we know that there is only one such path.

(ii) $\Rightarrow$ (iii):  $G$  is connected. Let us use induction on  $m$ .

Induction Basis:  $m = 0$ ,  $G$  is trivial and the statement is obvious.

Induction Hypothesis:  $m = n - 1$  when  $m \leq \ell$ . ( $\ell \geq 0$ )

Induction Statement:  $m = n - 1$  when  $m = \ell + 1$ .

Induction Statement Proof: Let  $e$  be an edge in  $G$ . Then  $G - e$  has  $\ell$  edges. If  $G - e$  is connected, then there exist two different paths between the end vertices of  $e$  so (ii) is false. Therefore,  $G - e$  has two components  $G_1$  and  $G_2$ . Let there be  $n_1$  vertices and  $m_1$  edges in  $G_1$ . Similarly, let there be  $n_2$  vertices and  $m_2$  edges in  $G_2$ . Then,

$$n = n_1 + n_2 \quad \text{and} \quad m = m_1 + m_2 + 1.$$

The Induction Hypothesis states that

$$m_1 = n_1 - 1 \quad \text{and} \quad m_2 = n_2 - 1,$$

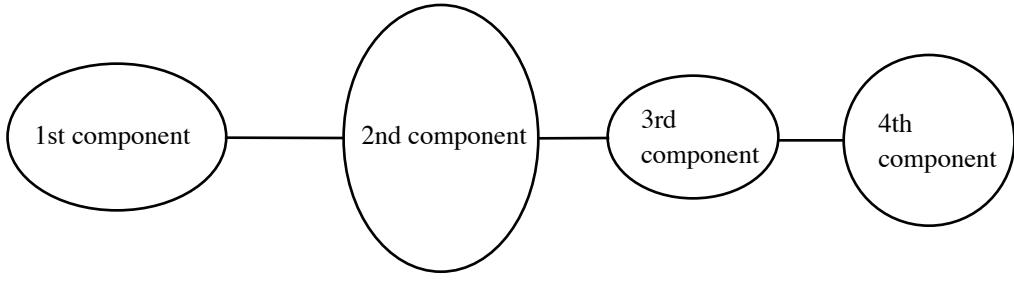
so  $m = n_1 + n_2 - 1 = n - 1$ .

(iii) $\Rightarrow$ (iv): Consider the counter hypothesis: There is a circuit in  $G$ . Let  $e$  be some edge in that circuit. Thus, there are  $n$  vertices and  $n - 2$  edges in the connected graph  $G - e$ . ✓<sup>1</sup>

(iv) $\Rightarrow$ (v): If  $G$  is circuitless, then there is at most one path between any two vertices (Theorem 1.6). If  $G$  has more than one component, then we will not get a circuit when we draw an edge between two different components. By adding edges, we can connect components without creating circuits:

---

<sup>1</sup>In a connected graph with  $n$  vertices, there are at least  $n - 1$  edges. (Theorem 1.4)



If we add  $k (\geq 1)$  edges, then (because (i) $\Rightarrow$ (iii))

$$m + k = n - 1 \quad (\checkmark \text{ because } m = n - 1).$$

So  $G$  is connected. When we add an edge between vertices that are not adjacent, we get only one circuit. Otherwise, we can remove an edge from one circuit so that other circuits will not be affected and the graph stays connected, in contradiction to (iii) $\Rightarrow$ (iv). Similarly, if we add a parallel edge or a loop, we get exactly one circuit.

(v) $\Rightarrow$ (i): Consider the counter hypothesis:  $G$  is not a tree, i.e. it is not connected. When we add edges as we did previously, we do not create any circuits (see figure).  $\checkmark$   $\square$

Since spanning trees are trees, Theorem 2.1 is also true for spanning trees.

**Theorem 2.2.** *A connected graph has at least one spanning tree.*

*Proof.* Consider the connected graph  $G$  with  $n$  vertices and  $m$  edges. If  $m = n - 1$ , then  $G$  is a tree. Since  $G$  is connected,  $m \geq n - 1$  (Theorem 1.4). We still have to consider the case  $m \geq n$ , where there is a circuit in  $G$ . We remove an edge  $e$  from that circuit.  $G - e$  is now connected. We repeat until there are  $n - 1$  edges. Then, we are left with a tree.  $\square$

**Remark.** *We can get a spanning tree of a connected graph by starting from an arbitrary sub-forest  $M$  (as we did previously). Since there is no circuit whose edges are all in  $M$ , we can remove those edges from the circuit which are not in  $M$ .*

By Theorem 2.1, the subgraph  $G_1$  of  $G$  with  $n$  vertices is a spanning tree of  $G$  (thus  $G$  is connected) if any three of the following four conditions hold:

1.  $G_1$  has  $n$  vertices.
2.  $G_1$  is connected.
3.  $G_1$  has  $n - 1$  edges.
4.  $G_1$  is circuitless.

Actually, conditions #3 and #4 are enough to guarantee that  $G_1$  is a spanning tree. If conditions #3 and #4 hold but  $G_1$  is not connected, then the components of  $G_1$  are trees and the number of edges in  $G_1$  would be

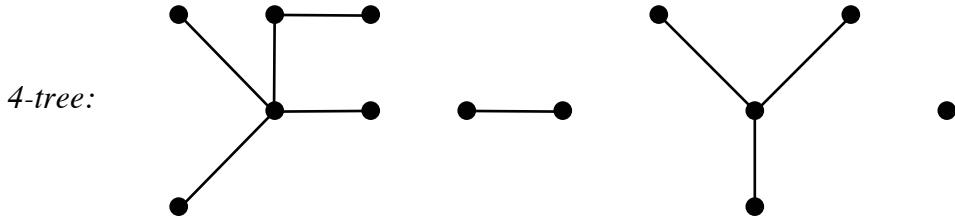
$$\text{number of vertices} - \text{number of components} < n - 1 \quad (\checkmark).$$

**Theorem 2.3.** If a tree is not trivial, then there are at least two pendant vertices.

*Proof.* If a tree has  $n(\geq 2)$  vertices, then the sum of the degrees is  $2(n - 1)$ . If every vertex has a degree  $\geq 2$ , then the sum will be  $\geq 2n$  ( $\checkmark$ ). On the other hand, if all but one vertex have degree  $\geq 2$ , then the sum would be  $\geq 1 + 2(n - 1) = 2n - 1$  ( $\checkmark$ ). (This also follows from Theorem 1.8 because a cut vertex of a tree is not a pendant vertex!)  $\square$

A forest with  $k$  components is sometimes called a *k-tree*. (So a 1-tree is a tree.)

**Example.**

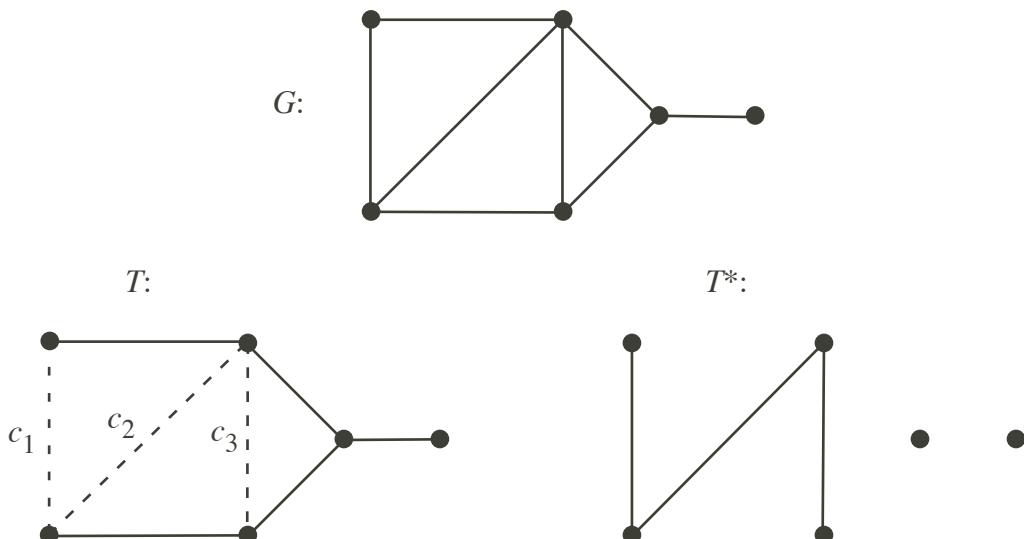


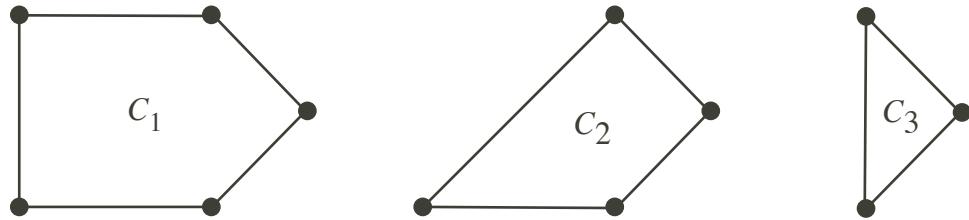
We use Theorem 2.1 to see that a graph with  $k$  components has a *spanning k-tree*, also known as a *spanning forest*, which has  $k$  components.

## 2.2 (Fundamental) Circuits and (Fundamental) Cut Sets

If the branches of the spanning tree  $T$  of a connected graph  $G$  are  $b_1, \dots, b_{n-1}$  and the corresponding links of the cospanning tree  $T^*$  are  $c_1, \dots, c_{m-n+1}$ , then there exists one and only one circuit  $C_i$  in  $T + c_i$  (which is the subgraph of  $G$  induced by the branches of  $T$  and  $c_i$ ) (Theorem 2.1). We call this circuit a *fundamental circuit*. Every spanning tree defines  $m - n + 1$  fundamental circuits  $C_1, \dots, C_{m-n+1}$ , which together form a *fundamental set of circuits*. Every fundamental circuit has exactly one link which is not in any other fundamental circuit in the fundamental set of circuits. Therefore, we can not write any fundamental circuit as a ring sum of other fundamental circuits in the same set. In other words, the fundamental set of circuits is linearly independent under the ring sum operation.

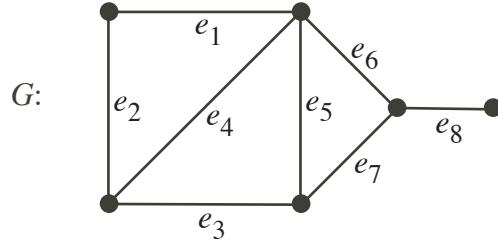
**Example.**



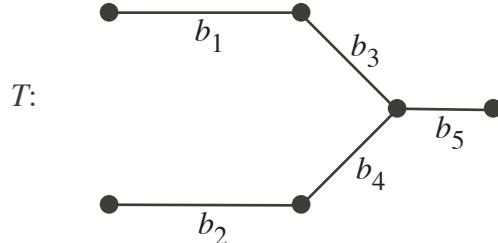


The graph \$T - b\_i\$ has two components \$T\_1\$ and \$T\_2\$. The corresponding vertex sets are \$V\_1\$ and \$V\_2\$. Then, \$\langle V\_1, V\_2 \rangle\$ is a cut of \$G\$. It is also a cut set of \$G\$ if we treat it as an edge set because \$G - \langle V\_1, V\_2 \rangle\$ has two components (result #1 p. 17). Thus, every branch \$b\_i\$ of \$T\$ has a corresponding cut set \$I\_i\$. The cut sets \$I\_1, \dots, I\_{n-1}\$ are also known as *fundamental cut sets* and they form a *fundamental set of cut sets*. Every fundamental cut set includes exactly one branch of \$T\$ and every branch of \$T\$ belongs to exactly one fundamental cut set. Therefore, every spanning tree defines a unique fundamental set of cut sets for \$G\$.

**Example.** (Continuing from the previous example) The graph



has the spanning tree



that defines these fundamental cut sets:

$$\begin{array}{lll} b_1 : \{e_1, e_2\} & b_2 : \{e_2, e_3, e_4\} & b_3 : \{e_2, e_4, e_5, e_6\} \\ b_4 : \{e_2, e_4, e_5, e_7\} & b_5 : \{e_8\} & \end{array}$$

Next, we consider some properties of circuits and cut sets:

- (a) Every cut set of a connected graph \$G\$ includes at least one branch from every spanning tree of \$G\$. (Counter hypothesis: Some cut set \$F\$ of \$G\$ does not include any branches of a spanning tree \$T\$. Then, \$T\$ is a subgraph of \$G - F\$ and \$G - F\$ is connected. ✓)
- (b) Every circuit of a connected graph \$G\$ includes at least one link from every cospanning tree of \$G\$. (Counter hypothesis: Some circuit \$C\$ of \$G\$ does not include any link of a cospanning tree \$T^\*\$. Then, \$T = G - T^\*\$ has a circuit and \$T\$ is not a tree. ✓)

**Theorem 2.4.** *The edge set  $F$  of the connected graph  $G$  is a cut set of  $G$  if and only if*

- (i)  *$F$  includes at least one branch from every spanning tree of  $G$ , and*
- (ii) *if  $H \subset F$ , then there is a spanning tree none of whose branches is in  $H$ .*

*Proof.* Let us first consider the case where  $F$  is a cut set. Then, (i) is true (previous proposition (a)). If  $H \subset F$  then  $G - H$  is connected and has a spanning tree  $T$ . This  $T$  is also a spanning tree of  $G$ . Hence, (ii) is true.

Let us next consider the case where both (i) and (ii) are true. Then  $G - F$  is disconnected. If  $H \subset F$  there is a spanning tree  $T$  none of whose branches is in  $H$ . Thus  $T$  is a subgraph of  $G - H$  and  $G - H$  is connected. Hence,  $F$  is a cut set.  $\square$

Similarly:

**Theorem 2.5.** *The subgraph  $C$  of the connected graph  $G$  is a circuit if and only if*

- (i)  *$C$  includes at least one link from every cospanning tree of  $G$ , and*
- (ii) *if  $D$  is a subgraph of  $C$  and  $D \neq C$ , then there exists a cospanning tree none of whose links is in  $D$ .*

*Proof.* Let us first consider the case where  $C$  is a circuit. Then,  $C$  includes at least one link from every cospanning tree (property (b) above) so (i) is true. If  $D$  is a proper subgraph of  $C$ , it obviously does not contain circuits, i.e. it is a forest. We can then supplement  $D$  so that it is a spanning tree of  $G$  (see remark on p. 22), i.e. some spanning tree  $T$  of  $G$  includes  $D$  and  $D$  does not include any link of  $T^*$ . Thus, (ii) is true.

Now we consider the case where (i) and (ii) are both true. Then, there has to be at least one circuit in  $C$  because  $C$  is otherwise a forest and we can supplement it so that it is a spanning tree of  $G$  (see remark on p. 22). We take a circuit  $C'$  in  $C$ . Since (ii) is true,  $C' \neq C$  is not true, because  $C'$  is a circuit and it includes a link from every cospanning tree (see property (b) above). Therefore,  $C = C'$  is a circuit.  $\square$

**Theorem 2.6.** *A circuit and a cut set of a connected graph have an even number of common edges.*

*Proof.* We choose a circuit  $C$  and a cut set  $F$  of the connected graph  $G$ .  $G - F$  has two components  $G_1 = (V_1, E_1)$  and  $G_2 = (V_2, E_2)$ . If  $C$  is a subgraph of  $G_1$  or  $G_2$ , then the theorem is obvious because they have no common edges. Let us assume that  $C$  and  $F$  have common edges. We traverse around a circuit by starting at some vertex  $v$  of  $G_1$ . Since we come back to  $v$ , there has to be an even number of edges of the cut  $\langle V_1, V_2 \rangle$  in  $C$ .  $\square$

The reader is advised to read the following several times:

**Theorem 2.7.** *A fundamental circuit corresponding to link  $c$  of the cospanning tree  $T^*$  of a connected graph is formed exactly by those branches of  $T$  whose corresponding fundamental cut set includes  $c$ .*

*Proof.* There exists a fundamental circuit  $C$  that corresponds to link  $c$  of  $T^*$ . The other edges  $b_1, \dots, b_k$  of  $C$  are branches of  $T$ . We denote  $I_i$  as the fundamental cut set that corresponds to branch  $b_i$ . Then,  $b_i$  is the only branch of  $T$  which is in both  $C$  and  $I_i$ . On the other hand,  $c$  is the only link of  $T^*$  in  $C$ . By Theorem 2.6, we know that the common edges of  $C$  and  $I_i$  are  $b_i$  and  $c$ , in other words,  $c$  is an edge of  $I_i$ . Then, we show that there is no  $c$  in the fundamental cut sets  $I_{k+1}, \dots, I_{n-1}$  that correspond to the branches  $b_{k+1}, \dots, b_{n-1}$  of  $T$ . For instance, if  $c$  were in  $I_{k+1}$ , then the fundamental cut set  $I_{k+1}$  and the circuit  $C$  would have exactly one common edge. ( $\checkmark$ ). So  $c$  is only in the fundamental cut sets  $I_1, \dots, I_k$ .  $\square$

The following is the corresponding theorem for fundamental cut sets:

**Theorem 2.8.** *The fundamental cut set corresponding to branch  $b$  of the spanning tree  $T$  of a connected graph consists exactly of those links of  $T^*$  whose corresponding fundamental circuit includes  $b$ .*

*Proof.* Let  $I$  be a fundamental cut set that corresponds to the branch  $b$  of  $T$ . Other edges  $c_1, \dots, c_k$  of  $I$  are links of  $T^*$ . Let  $C_i$  denote the fundamental circuit that corresponds to  $c_i$ . Then,  $c_i$  is the only link of  $T^*$  in both  $I$  and  $C_i$ . On the other hand,  $b$  is the only branch of  $T$  in  $I$ . By Theorem 2.6, the common edges of  $I$  and  $C_i$  are  $b$  and  $c_i$ , in other words,  $b$  is an edge of  $C_i$ . Then, we show that the fundamental circuits  $C_{k+1}, \dots, C_{m-n+1}$  corresponding to the links  $c_{k+1}, \dots, c_{m-n+1}$  do not include  $b$ . For example, if  $b$  were in  $C_{k+1}$ , then the fundamental circuit  $C_{k+1}$  and the cut set  $I$  would have exactly one common edge (✓). Hence, the branch  $b$  is only in fundamental circuits  $C_1, \dots, C_k$ .  $\square$

From the results, we can see the duality between cut sets and circuits of a graph: The theorems for cut sets can generally be converted to dual theorems for circuits and vice versa. Usually, we just need to change some of the key terminologies to their duals in the theorems and proofs. In particular, we take advantage of this dualism for dealing with matroids (see Chapter 7).

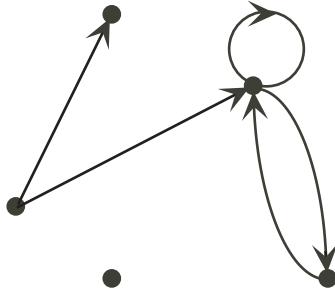
# Chapter 3

## Directed Graphs

### 3.1 Definition

Intuitively, a *directed graph* or *digraph* is formed by vertices connected by *directed edges* or *arcs*.<sup>1</sup>

**Example.**



Formally, a digraph is a pair  $(V, E)$ , where  $V$  is the vertex set and  $E$  is the set of vertex pairs as in "usual" graphs. The difference is that now the elements of  $E$  are ordered pairs: the arc from vertex  $u$  to vertex  $v$  is written as  $(u, v)$  and the other pair  $(v, u)$  is the opposite direction arc. We also have to keep track of the multiplicity of the arc (direction of a loop is irrelevant). We can pretty much use the same notions and results for digraphs from Chapter 1. However:

1. Vertex  $u$  is the *initial vertex* and vertex  $v$  is the *terminal vertex* of the arc  $(u, v)$ . We also say that the arc is *incident out of  $u$*  and *incident into  $v$* .
2. The *out-degree* of the vertex  $v$  is the number of arcs out of it (denoted  $d^+(v)$ ) and the *in-degree* of  $v$  is the number of arcs going into it (denoted  $d^-(v)$ ).
3. In the *directed walk (trail, path or circuit)*,

$$v_{i_0}, e_{j_1}, v_{i_1}, e_{j_2}, \dots, e_{j_k}, v_{i_k}$$

$v_{i_\ell}$  is the initial vertex and  $v_{i_{\ell-1}}$  is the terminal vertex of the arc  $e_{j_\ell}$ .

4. When we treat the graph  $(V, E)$  as a usual undirected graph, it is the *underlying undirected graph* of the digraph  $G = (V, E)$ , denoted  $G_u$ .

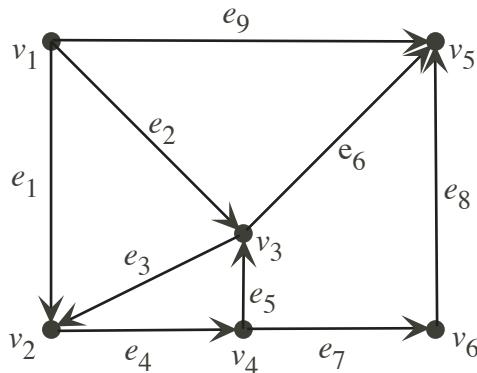
---

<sup>1</sup>This is not a standard terminology. We will however call directed edges arcs in the sequel.

5. Digraph  $G$  is *connected* if  $G_u$  is connected. The *components* of  $G$  are the directed subgraphs of  $G$  that correspond to the components of  $G_u$ . The vertices of  $G$  are connected if they are connected in  $G_u$ . Other notions for undirected graphs can be used for digraphs as well by dealing with the underlying undirected graph.
6. Vertices  $u$  and  $v$  are *strongly connected* if there is a directed  $u-v$  path and also a directed  $v-u$  path in  $G$ .
7. Digraph  $G$  is *strongly connected* if every pair of vertices is strongly connected. By convention, the trivial graph is strongly connected.
8. A *strongly connected component*  $H$  of the digraph  $G$  is a directed subgraph of  $G$  (not a null graph) such that  $H$  is strongly connected, but if we add any vertices or arcs to it, then it is not strongly connected anymore.

Every vertex of the digraph  $G$  belongs to one strongly connected component of  $G$  (compare to Theorem 1.3). However, an arc does not necessarily belong to any strongly connected component of  $G$ .

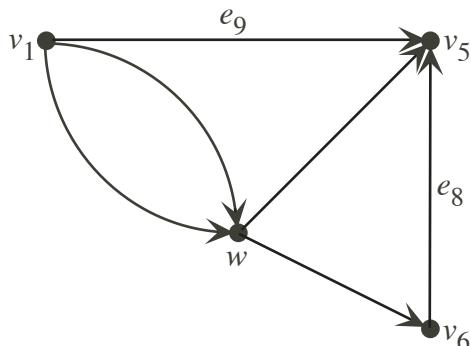
**Example.** For the digraph  $G$



the strongly connected components are  $(\{v_1\}, \emptyset)$ ,  $(\{v_2, v_3, v_4\}, \{e_3, e_4, e_5\})$ ,  $(\{v_5\}, \emptyset)$  and  $(\{v_6\}, \emptyset)$ .

The *condensed graph*  $G_c$  of the digraph  $G$  is obtained by contracting all the arcs in every strongly connected component.

**Example.** (Continuing from the previous example) The condensed graph is



## 3.2 Directed Trees

A directed graph is *quasi-strongly connected* if one of the following conditions holds for every pair of vertices  $u$  and  $v$ :

- (i)  $u = v$  or
- (ii) there is a directed  $u-v$  path in the digraph or
- (iii) there is a directed  $v-u$  path in the digraph or
- (iv) there is a vertex  $w$  so that there is a directed  $w-u$  path and a directed  $w-v$  path.

**Example.** (Continuing from the previous example) The digraph  $G$  is quasi-strongly connected.

Quasi-strongly connected digraphs are connected but not necessarily strongly connected.

The vertex  $v$  of the digraph  $G$  is a *root* if there is a directed path from  $v$  to every other vertex of  $G$ .

**Example.** (Continuing from the previous example) The digraph  $G$  only has one root,  $v_1$ .

**Theorem 3.1.** A digraph has at least one root if and only if it is quasi-strongly connected.

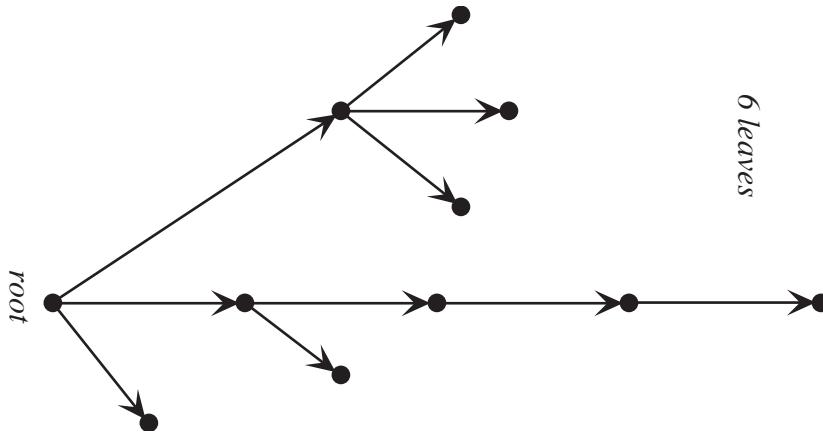
*Proof.* If there is a root in the digraph, it follows from the definition that the digraph is quasi-strongly connected.

Let us consider a quasi-strongly connected digraph  $G$  and show that it must have at least one root. If  $G$  is trivial, then it is obvious. Otherwise, consider the vertex set  $V = \{v_1, \dots, v_n\}$  of  $G$  where  $n \geq 2$ . The following process shows that there must be a root:

1. Set  $P \leftarrow V$ .
2. If there is a directed  $u-v$  path between two distinct vertices  $u$  and  $v$  in  $P$ , then we remove  $v$  from  $P$ . Equivalently, we set  $P \leftarrow P - \{v\}$ . We repeat this step as many times as possible.
3. If there is only one vertex left in  $P$ , then it is the root. For other cases, there are at least two distinct vertices  $u$  and  $v$  in  $P$  and there is no directed path between them in either direction. Since  $G$  is quasi-strongly connected, from condition (iv) it follows that there is a vertex  $w$  and a directed  $w-u$  path as well as a directed  $w-v$  path. Since  $u$  is in  $P$ ,  $w$  can not be in  $P$ . We remove  $u$  and  $v$  from  $P$  and add  $w$ , i.e. we set  $P \leftarrow P - \{u, v\}$  and  $P \leftarrow P \cup \{w\}$ . Go back to step #2.
4. Repeat as many times as possible.

Every time we do this, there are fewer and fewer vertices in  $P$ . Eventually, we will get a root because there is a directed path from some vertex in  $P$  to every vertex we removed from  $P$ .  $\square$

The digraph  $G$  is a *tree* if  $G_u$  is a tree. It is a *directed tree* if  $G_u$  is a tree and  $G$  is quasi-strongly connected, i.e. it has a root. A *leaf* of a directed tree is a vertex whose out-degree is zero.

**Example.**

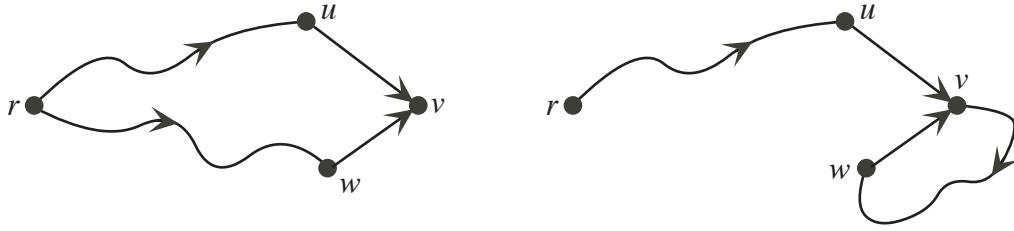
**Theorem 3.2.** For the digraph  $G$  with  $n > 1$  vertices, the following are equivalent:

- (i)  $G$  is a directed tree.
- (ii)  $G$  is a tree with a vertex from which there is exactly one directed path to every other vertex of  $G$ .
- (iii)  $G$  is quasi-strongly connected but  $G - e$  is not quasi-strongly connected for any arc  $e$  in  $G$ .
- (iv)  $G$  is quasi-strongly connected and every vertex of  $G$  has an in-degree of 1 except one vertex whose in-degree is zero.
- (v) There are no circuits in  $G$  (i.e. not in  $G_u$ ) and every vertex of  $G$  has an in-degree of 1 except one vertex whose in-degree is zero.
- (vi)  $G$  is quasi-strongly connected and there are no circuits in  $G$  (i.e. not in  $G_u$ ).

*Proof.* (i) $\Rightarrow$ (ii): If  $G$  is a directed tree, then there is a root. This implies that there is a directed path from the root to every other vertex in  $G$  (but not more than one path since  $G_u$  is a tree).

(ii) $\Rightarrow$ (iii): If (ii) is true, then  $G$  obviously is quasi-strongly connected. We will prove by contradiction by considering the counter hypothesis: There is an arc  $e$  in  $G$  such that  $G - e$  is quasi-strongly connected. The arc  $e$  is not a loop because  $G$  is a directed tree. Let  $u$  and  $v$  be the two different end vertices of  $e$ . There does not exist a directed  $u-v$  path or a directed  $v-u$  path in  $G - e$  (otherwise  $G_u$  would have a circuit). Therefore, there is a vertex  $w$  and a directed  $w-u$  path as well as a directed  $w-v$  path. However, this leads to the existence of two directed  $w-u$  paths or two directed  $w-v$  paths in  $G$  depending on the direction of the arc  $e$ . Then, there is a circuit in the tree  $G_u$ . ( $\checkmark$  by Theorem 1.6).

(iii) $\Rightarrow$ (iv): If  $G$  quasi-strongly connected, then it has a root  $r$  (Theorem 3.1) so that the in-degrees of other vertices are  $\geq 1$ . We start by considering the counter hypothesis: There exists a vertex  $v \neq r$  and  $d^-(v) > 1$ . Then,  $v$  is the terminal vertex of two distinct arcs  $(u, v)$  and  $(w, v)$ . If there were a loop  $e$  in  $G$ , then  $G - e$  would be quasi-strongly connected ( $\checkmark$ ). Thus,  $u \neq w$  with  $w \neq v$ . Now, there are two distinct directed trails from  $r$  to  $v$ . The first one includes  $(u, v)$  and the second one includes  $(w, v)$ . We have two possible cases:



In the digraph on the left, the paths  $r-u$  and  $r-w$  do not include the arcs  $(u, v)$  and  $(w, v)$ . Both  $G - (u, v)$  and  $G - (w, v)$  are quasi-strongly connected. In the digraph on the right, the  $r-u$  path includes the arc  $(w, v)$  or (as in the figure) the  $r-w$  path includes the arc  $(u, v)$ . In either case, only one of  $G - (u, v)$  and  $G - (w, v)$  is quasi-strongly connected because the root is  $r$  (Theorem 3.1). (✓) We still have to show that  $d^-(r) = 0$ . Let us consider the counter hypothesis:  $d^-(r) \geq 1$ . Then,  $r$  is the terminal vertex of some arc  $e$ . However, the tree  $G - e$  is then quasi-strongly connected since  $r$  is its root (Theorem 3.1). (✓)

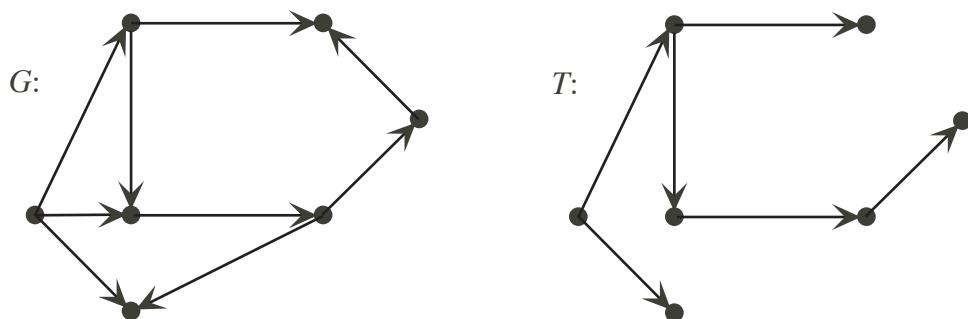
(iv) $\Rightarrow$ (v): If (iv) is true, then it is enough to show that there are no circuits in  $G_u$ . The sum of in-degrees of all the vertices in  $G$  is  $n - 1$  and the sum of out-degrees of all the vertices in  $G$  is also  $n - 1$ , i.e. there are  $n - 1$  arcs in  $G$ . Since  $G$  is quasi-strongly connected, it is connected and it is a tree (Theorem 2.1). Therefore, there are no circuits in  $G_u$ .

(v) $\Rightarrow$ (vi): If we assume that (v) is true, then there are  $n - 1$  arcs in  $G$  (compare to the previous proof). By Theorem 2.1,  $G$  is a tree. We denote by  $r$  the vertex satisfying condition (v). By Theorem 2.1, we see that there is exactly one path to any other vertex of  $G$  from  $r$ . These paths are also directed. Otherwise,  $d^-(r) \geq 1$  or the in-degree of some vertex on that path is  $> 1$  or the in-degree of some other vertex other than  $r$  on that path is zero. Hence,  $r$  is a root and  $G$  is quasi-strongly connected (Theorem 3.1).

(vi) $\Rightarrow$ (i): If  $G$  is quasi-strongly connected, then it has a root (Theorem 3.1). Since  $G$  is connected and there are no circuits in  $G$ , it is a tree.  $\square$

A directed subgraph  $T$  of the digraph  $G$  is a *directed spanning tree* if  $T$  is a directed tree and  $T$  includes every vertex of  $G$ .

### Example.



**Theorem 3.3.** A digraph has a directed spanning tree if and only if it is quasi-strongly connected.

*Proof.* If the digraph  $G$  has a directed spanning tree  $T$ , then the root of  $T$  is also a root for  $G$  and it is quasi-strongly connected (Theorem 3.1).

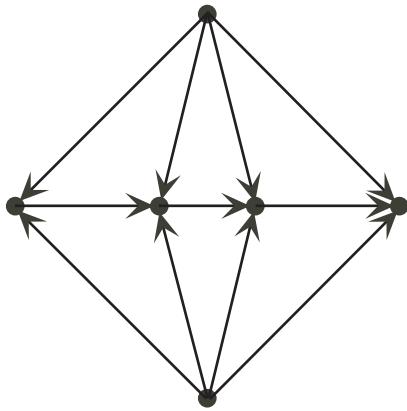
We now assume that  $G$  is quasi-strongly connected and show that it has a directed spanning tree. If  $G$  is a directed tree, then it is obvious. Otherwise, from Theorem 3.2, we know that there

is an arc  $e$  in  $G$  so that if we remove  $e$ ,  $G$  remains quasi-strongly connected. We systematically remove these kind of arcs until we get a directed tree. (Compare to the proof for Theorem 2.2)  $\square$

### 3.3 Acyclic Directed Graphs

A directed graph with at least one directed circuit is said to be *cyclic*. A directed graph is *acyclic* otherwise. Obviously, directed trees are acyclic but the reverse implication is not true.

**Example.** *The digraph*



is acyclic but it is not a directed tree.

**Theorem 3.4.** *In an acyclic digraph, there exist at least one source (a vertex whose in-degree is zero) and at least one sink (a vertex whose out-degree is zero).*

*Proof.* Let  $G$  be an acyclic digraph. If  $G$  has no arcs, then it is obvious. Otherwise, let us consider the directed path

$$v_{i_0}, e_{j_1}, v_{i_1}, e_{j_2}, \dots, e_{j_k}, v_{i_k},$$

which has the maximum path length  $k$ . Since  $G$  is acyclic,  $v_{i_0} \neq v_{i_k}$ . If  $(v, v_{i_0})$  is an arc, then one of the following is true:

- $v \neq v_{i_t}$  for every value of  $t = 0, \dots, k$ . Then,

$$v, (v, v_{i_0}), v_{i_0}, e_{j_1}, v_{i_1}, e_{j_2}, \dots, e_{j_k}, v_{i_k}$$

is a directed path with length  $k + 1$ .  $\checkmark$

- $v = v_{i_t}$  for some value of  $t$ . We choose the smallest such  $t$ . Then,  $t > 0$  because there are no loops in  $G$  and

$$v_{i_0}, e_{j_1}, v_{i_1}, e_{j_2}, \dots, e_{j_t}, v_{i_t}, (v, v_{i_0}), v_{i_0}$$

is a directed circuit.  $\checkmark$

Hence,  $d^-(v_{i_0}) = 0$ . Using a similar technique, we can show that  $d^+(v_{i_k}) = 0$  as well.  $\square$

If  $G = (V, E)$  is a digraph with  $n$  vertices, then a labeling of the vertices with an injective function  $\alpha : V \rightarrow \{1, \dots, n\}$  which satisfies the condition  $\alpha(u) < \alpha(v)$  whenever  $(u, v)$  is an arc in  $G$  is known as *topological sorting*.

**Theorem 3.5.** *We can sort the vertices of a digraph topologically if and only if the graph is acyclic.*

*Proof.* If the digraph is cyclic, then obviously we can not sort the vertices topologically.

If the digraph  $G$  is acyclic, then we can sort the vertices in the following manner:<sup>2</sup>

1. We choose a vertex  $v$  which is a sink. It exists by Theorem 3.4. We set  $\alpha(v) \leftarrow n$ ,  $G \leftarrow G - v$  and  $n \leftarrow n - 1$ .
2. If there is just one vertex  $v$  in  $G$ , set  $\alpha(v) \leftarrow 1$ . Otherwise, go back to step #1.  $\square$

---

<sup>2</sup>This is known as *Marimont's Algorithm*. The algorithm itself contains other items, too. The original reference is MARIMONT, R.B.: A New Method of Checking the Consistency of Precedence Matrices. *Journal of the Association for Computing Machinery* **6** (1959), 164–171.

# Chapter 4

## Matrices and Vector Spaces of Graphs

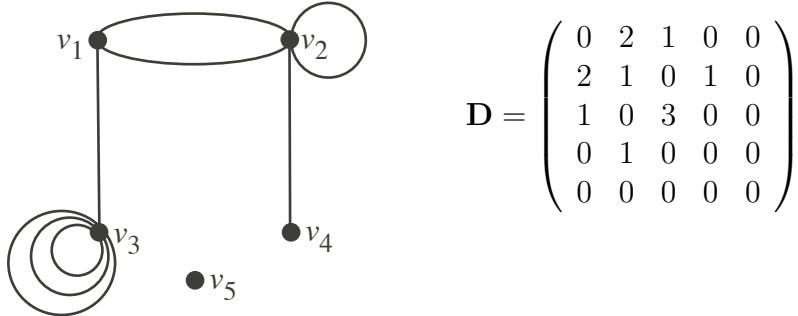
### 4.1 Matrix Representation of Graphs

The *adjacency matrix* of the graph  $G = (V, E)$  is an  $n \times n$  matrix  $\mathbf{D} = (d_{ij})$ , where  $n$  is the number of vertices in  $G$ ,  $V = \{v_1, \dots, v_n\}$  and

$$d_{ij} = \text{number of edges between } v_i \text{ and } v_j.$$

In particular,  $d_{ij} = 0$  if  $(v_i, v_j)$  is not an edge in  $G$ . The matrix  $\mathbf{D}$  is symmetric, i.e.  $\mathbf{D}^T = \mathbf{D}$ .

**Example.**

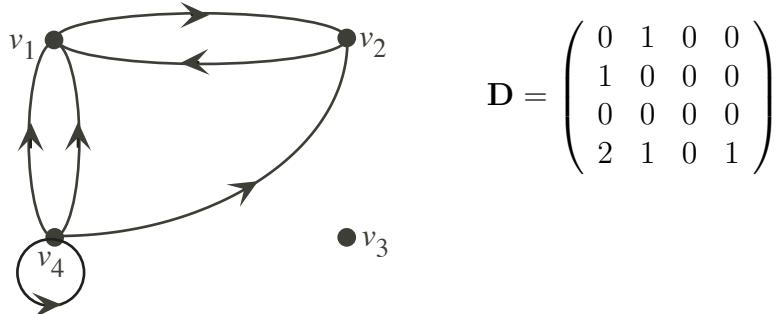


Obviously, an adjacency matrix defines a graph completely up to an isomorphism.

The adjacency matrix of a directed graph  $G$  is  $\mathbf{D} = (d_{ij})$ , where

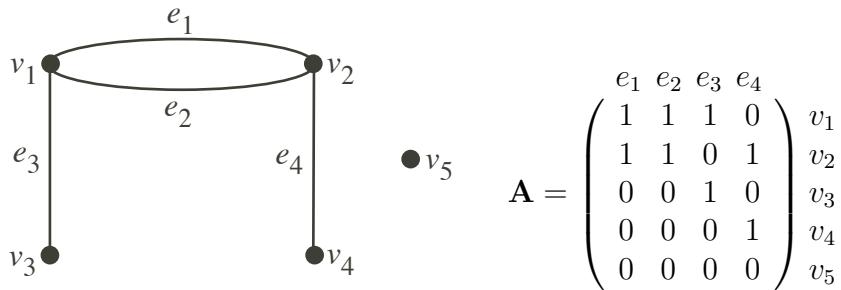
$$d_{ij} = \text{number of arcs that come out of vertex } v_i \text{ and go into vertex } v_j.$$

**Example.**



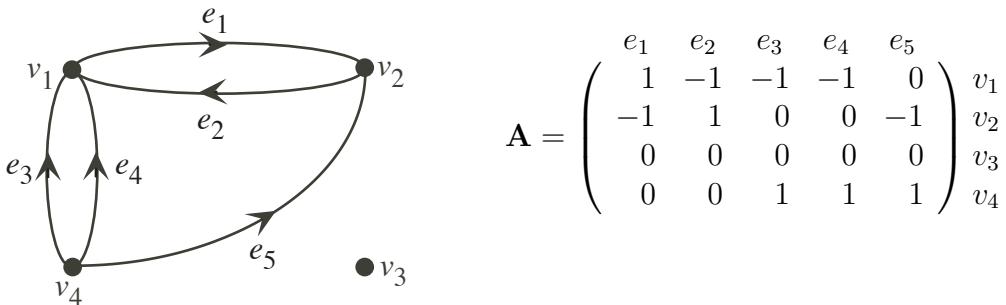
The *all-vertex incidence matrix* of a non-empty and loopless graph  $G = (V, E)$  is an  $n \times m$  matrix  $\mathbf{A} = (a_{ij})$ , where  $n$  is the number of vertices in  $G$ ,  $m$  is the number of edges in  $G$  and

$$a_{ij} = \begin{cases} 1 & \text{if } v_i \text{ is an end vertex of } e_j \\ 0 & \text{otherwise.} \end{cases}$$

**Example.**

The *all-vertex incidence matrix* of a non-empty and loopless directed graph  $G$  is  $\mathbf{A} = (a_{ij})$ , where

$$a_{ij} = \begin{cases} 1 & \text{if } v_i \text{ is the initial vertex of } e_j \\ -1 & \text{if } v_i \text{ is the terminal vertex of } e_j \\ 0 & \text{otherwise.} \end{cases}$$

**Example.**

Since every column of an all-vertex incidence matrix contains exactly two non-zero numbers, two ones, we can remove a row and still have enough information to define the graph. The *incidence matrix* of a graph is obtained by removing a row from the all-vertex incidence matrix. It is not unique because there are  $n$  possible rows to remove. The vertex corresponding to the row removed is called the *reference vertex*.

Similarly, every column in the all-vertex incidence matrix of a digraph contains exactly two non-zero numbers, +1 and -1. We can remove a row from the all-vertex incidence matrix and obtain the *incidence matrix*. Notice that the rows of an all-vertex incidence matrix are linearly dependent because the sum of rows is a zero vector.

**Theorem 4.1.** *The determinant of an incidence matrix of a nontrivial tree is  $\pm 1$ , regardless of whether the tree is a directed graph or not.*

*Proof.* We use induction on  $n$ , the number of vertices in the tree.

Induction Basis:  $n = 2$  and it is obvious.

Induction Hypothesis: The theorem is true for  $n \leq k$ . ( $k \geq 2$ )

Induction Statement: The theorem is true for  $n = k + 1$ .

Induction Statement Proof: Let  $T$  be a tree which has  $k + 1$  vertices and let  $\mathbf{A}$  be an (arbitrary) incidence matrix of  $T$ .  $T$  has at least two pendant vertices (Theorem 2.3). We choose a pendant vertex  $v_i$  which is not the reference vertex of  $\mathbf{A}$  and the edge  $e_t$  which is incident on  $v_i$ . Then,

$$a_{it} = (\pm)1 \quad \text{and} \quad a_{ij} = 0, \text{ when } j \neq t.$$

We expand the determinant of  $|\mathbf{A}|$  by the  $i^{\text{th}}$  row:

$$|\mathbf{A}| = (\pm)(-1)^{i+t} |\mathbf{A}'|,$$

where  $\mathbf{A}'$  is the minor corresponding to  $a_{it}$ . We write  $T' = T - v_i$  which is also a tree ( $v_i$  is a pendant vertex). We use the induction hypothesis to get  $|\mathbf{A}'| = \pm 1$  because  $\mathbf{A}'$  is obviously an incidence matrix of  $T'$ .  $\square$

**Corollary.** *If the digraph  $G$  has no loops, then the rank of its all-vertex incidence matrix is  $\rho(G)$ .*

*Proof.* If we rearrange the rows or columns of the all-vertex incidence matrix, the rank of the matrix will not change. Let us rearrange the vertices and arcs to group them by components. Then, the all-vertex incidence matrix is a block diagonal matrix in which each block is an all-vertex incidence matrix of a component.

$$\begin{pmatrix} & \boxed{1^{\text{st}} \text{ compo-} \\ \text{nent}} & & & & \\ & & \boxed{2^{\text{nd}} \text{ compo-} \\ \text{nent}} & & & \\ & & & \ddots & & \\ & \mathbf{O} & & & & \\ & & & & \boxed{k^{\text{th}} \text{ compo-} \\ \text{nent}} & \end{pmatrix}$$

We denote  $n_i$  as the number of vertices in the  $i^{\text{th}}$  component. Every component has a spanning tree whose incidence matrix has a nonzero determinant by Theorem 4.1, i.e. the matrix is not singular. The all-vertex incidence matrix of the  $i^{\text{th}}$  component is obtained by adding columns and one row to an incidence matrix of the corresponding spanning tree. The row added is linearly dependent of other rows so that the rank of this matrix is the same as the rank of the incidence matrix ( $= n_i - 1$ ). Notice that in the special case when a component is trivial, the rank is zero  $= 1 - 1$ . Therefore,

$$\begin{aligned} \text{rank of } \mathbf{A} &= \text{sum of the ranks of the components} \\ &= (n_1 - 1) + \cdots + (n_k - 1) \\ &= \underbrace{n_1 + \cdots + n_k}_{= n} - k = \rho(G). \end{aligned} \quad \square$$

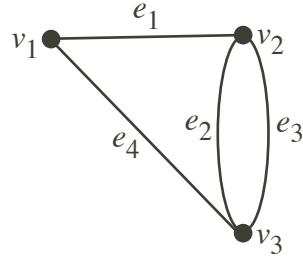
**Remark.** *From this proof, we can also get a basis for the row space and the column space of the all-vertex incidence matrix. The columns corresponding to the branches of the spanning forest of  $G$  are a basis of the column space. We can get a basis of the row space by removing one row out of each component block.*

## 4.2 Cut Matrix

If all the cuts of a nontrivial and loopless graph  $G = (V, E)$  are  $I_1, \dots, I_t$ , then the *cut matrix* of  $G$  is a  $t \times m$  matrix  $\mathbf{Q} = (q_{ij})$ , where  $m$  is the number of edges in  $G$  and

$$q_{ij} = \begin{cases} 1 & \text{if } e_j \in I_i \text{ (the cut is interpreted as an edge set)} \\ 0 & \text{otherwise.} \end{cases}$$

**Example.** For the graph



the cuts are  $I_1 = \{e_1, e_4\}$ ,  $I_2 = \{e_2, e_3, e_4\}$  and  $I_3 = \{e_1, e_2, e_3\}$ . The cut matrix is

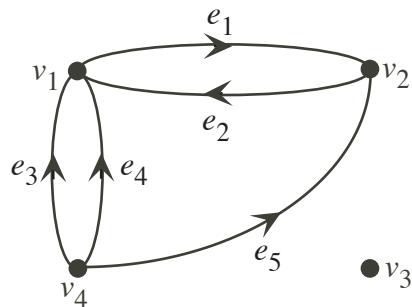
$$\mathbf{Q} = \begin{pmatrix} e_1 & e_2 & e_3 & e_4 \\ 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 \end{pmatrix} \begin{matrix} I_1 \\ I_2 \\ I_3 \end{matrix}$$

**Remark.** If the graph has  $n$  vertices, then it has  $\frac{1}{2}(2^n - 2) = 2^{n-1} - 1$  cuts. Usually, there are not this many distinct edge sets. For the cut matrix, we only take one cut corresponding to an edge set so that there would not be repeated rows. Even so, there are usually too many rows.

If  $G$  is a nontrivial and loopless digraph, then we assign an arbitrary direction to every cut  $\langle V_1, V_2 \rangle$ : the *orientation* of  $\langle V_1, V_2 \rangle$  is from  $V_1$  to  $V_2$ . In other words, we consider *oriented cuts* and we pick only one direction from the two possibilities. Then, the *cut matrix*  $\mathbf{Q} = (q_{ij})$  is

$$q_{ij} = \begin{cases} 1 & \text{if } e_j \in I_i \text{ and they are in the same direction} \\ -1 & \text{if } e_j \in I_i \text{ and they are in opposite directions} \\ 0 & \text{otherwise.} \end{cases}$$

**Example.** For the digraph



the different cuts (interpreted as edge sets) are  $I_1 = \{e_1, e_2, e_3, e_4\}$  (in the direction of  $e_1$ ),  $I_2 = \{e_3, e_4, e_5\}$  (in the direction of  $e_3$ ),  $I_3 = \{e_1, e_2, e_5\}$  (in the direction of  $e_1$ ) and  $I_4 = \emptyset$ . The cut matrix is

$$\mathbf{Q} = \begin{pmatrix} e_1 & e_2 & e_3 & e_4 & e_5 \\ 1 & -1 & -1 & -1 & 0 \\ 0 & 0 & 1 & 1 & 1 \\ 1 & -1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix} \begin{matrix} I_1 \\ I_2 \\ I_3 \\ I_4 \end{matrix}$$

Since  $\langle \{v\}, V - \{v\} \rangle$  is a cut for every vertex  $v$ , rows of the all-vertex incidence matrix are rows of  $\mathbf{Q}$ . If we are dealing with directed graphs, then these rows may have to be multiplied by  $-1$ .

**Theorem 4.2.** *Every row of the cut matrix of a digraph can be expressed in two different ways as a linear combination of the rows of the all-vertex incidence matrix. The non-zero coefficients are either all = +1 or all = -1.*

*Proof.* Let  $\mathbf{Q}$  be the cut matrix of a digraph  $G = (V, E)$  and let  $\mathbf{A}$  be the all-vertex incidence matrix. Let  $\langle V_1, V_2 \rangle$  (note that it is oriented) be the cut corresponding to the  $i^{\text{th}}$  row of  $\mathbf{Q}$ . Reindexing if needed, we can assume that

$$V_1 = \{v_1, \dots, v_r\} \quad \text{and} \quad V_2 = \{v_{r+1}, \dots, v_n\}.$$

We write

$$\mathbf{q}_i = i^{\text{th}} \text{ row of } \mathbf{Q} \quad \text{and} \quad \mathbf{a}_t = t^{\text{th}} \text{ row of } \mathbf{A}.$$

We show that

$$\mathbf{q}_i = \sum_{t=1}^r \mathbf{a}_t = - \sum_{t=r+1}^n \mathbf{a}_t,$$

which proves the theorem. Let  $(v_p, v_q) = e_k$  be the  $k^{\text{th}}$  arc of  $G$ . Then,

$$\begin{aligned} a_{pk} &= k^{\text{th}} \text{ element of the vector } \mathbf{a}_p = 1, \\ a_{qk} &= k^{\text{th}} \text{ element of the vector } \mathbf{a}_q = -1 \end{aligned}$$

and

$$a_{jk} = 0 \quad \text{if} \quad j \neq p, q.$$

We get four cases:

- $v_p \in V_1$  and  $v_q \in V_2$ : Now  $p \leq r$  and  $q \geq r+1$  so  $q_{ik} = 1$  and

$$q_{ik} = \sum_{t=1}^r a_{tk} = - \sum_{t=r+1}^n a_{tk}.$$

- $v_p \in V_2$  and  $v_q \in V_1$ : Now  $p \geq r+1$  and  $q \leq r$  so  $q_{ik} = -1$  and

$$q_{ik} = \sum_{t=1}^r a_{tk} = - \sum_{t=r+1}^n a_{tk}.$$

- $v_p \in V_1$  and  $v_q \in V_1$ : Now  $p \leq r$  and  $q \leq r$  so  $q_{ik} = 0$  and

$$q_{ik} = \sum_{t=1}^r a_{tk} = - \underbrace{a_{r+1,k}}_{=0} - \cdots - \underbrace{a_{nk}}_{=0}.$$

- $v_p \in V_2$  and  $v_q \in V_2$ : Now  $p \geq r+1$  and  $q \geq r+1$  so  $q_{ik} = 0$  and

$$q_{ik} = \underbrace{a_{1k}}_{=0} + \cdots + \underbrace{a_{rk}}_{=0} = - \sum_{t=r+1}^n a_{tk}.$$

The statements above are valid for every  $k$ .  $\square$

**Example.** (Continuing from the previous example) The corresponding row of  $I_1$  is

$$(1, -1, -1, -1, 0) = (1, -1, -1, -1, 0) = -(-1, 1, 0, 0, -1) - (0, 0, 0, 0, 0) - (0, 0, 1, 1, 1).$$

**Corollary.** The rank of the cut matrix of a digraph  $G$  is  $\rho(G)$ .

*Proof.* The all-vertex incidence matrix  $\mathbf{A}$  of  $G$  is also a submatrix of the cut matrix  $\mathbf{Q}$  of  $G$ . Then, (by Corollary of Theorem 4.1)

$$\text{rank}(\mathbf{Q}) \geq \text{rank}(\mathbf{A}) = \rho(G).$$

On the other hand, by Theorem 4.2, every row of  $\mathbf{Q}$  can be expressed as a linear combination of the rows of  $\mathbf{A}$ . Therefore,

$$\text{rank}(\mathbf{Q}) = \text{rank}(\mathbf{A}) = \rho(G).$$

$\square$

Another consequence is that the cut matrix  $\mathbf{Q}$  can be expressed as

$$\mathbf{Q} = \mathbf{A}_1 \mathbf{A},$$

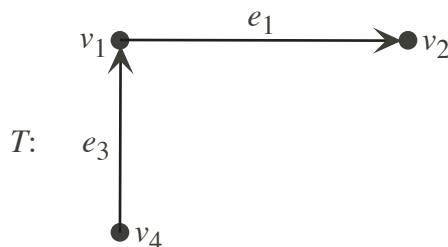
where the elements of  $\mathbf{A}_1$  are 0 or  $\pm 1$ . In addition, the matrix  $\mathbf{A}_1$  can be constructed from the process in the proof of Theorem 4.2.

If the graph  $G$  is connected, then it has a spanning tree  $T$  and an associated fundamental cut set. The fundamental cut sets are also cuts (when cuts are interpreted as edge sets). Therefore, the cut matrix  $\mathbf{Q}$  of  $G$  has a submatrix  $\mathbf{Q}_f$  that corresponds to these fundamental cut sets. This matrix is called the *fundamental cut set matrix*. Similarly, the connected digraph  $G$  has a fundamental cut set matrix: if we interpret a fundamental cut set as a set, then the direction of the cut is chosen to be the same as the direction of the corresponding branch of  $T$ . If we rearrange the edges of  $G$  so that we have the branches first and sort the fundamental cut sets in the same order, then we get the fundamental cut set matrix in the form

$$\mathbf{Q}_f = (\mathbf{I}_{n-1} \mid \mathbf{Q}_{fc}),$$

where  $\mathbf{I}_{n-1}$  is the identity matrix with  $n - 1$  rows. The rank of  $\mathbf{Q}_f$  is thus  $n - 1 = \rho(G)$ .

**Example.** (Continuing from the previous example) We left out vertex  $v_3$  so we get a connected digraph. We choose the spanning tree



The fundamental cut sets are  $I_2 = \{e_3, e_4, e_5\}$  (in the direction of  $e_3$ ) and  $I_3 = \{e_1, e_2, e_5\}$  (in the direction of  $e_1$ ). Then,

$$\mathbf{Q}_f = \left( \begin{array}{cc|ccccc} e_1 & e_3 & e_2 & e_4 & e_5 \\ 1 & 0 & -1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 1 \end{array} \right) I_3 \\ I_2$$

and

$$\mathbf{Q} = \left( \begin{array}{ccccc} e_1 & e_2 & e_3 & e_4 & e_5 \\ 1 & -1 & -1 & -1 & 0 \\ 0 & 0 & 1 & 1 & 1 \\ 1 & -1 & 0 & 0 & 1 \end{array} \right) \leftarrow \leftarrow$$

### 4.3 Circuit Matrix

We consider a loopless graph  $G = (V, E)$  which contains circuits. We enumerate the circuits of  $G$ :  $C_1, \dots, C_\ell$ . The *circuit matrix* of  $G$  is an  $\ell \times m$  matrix  $\mathbf{B} = (b_{ij})$  where

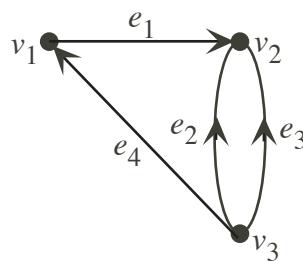
$$b_{ij} = \begin{cases} 1 & \text{if the arc } e_j \text{ is in the circuit } C_i \\ 0 & \text{otherwise} \end{cases}$$

(as usual,  $E = \{e_1, \dots, e_m\}$ ).

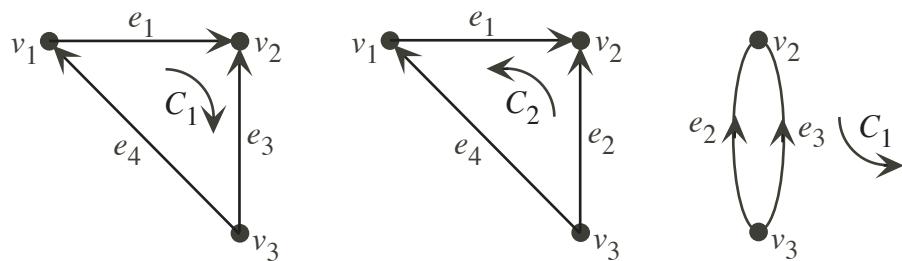
The circuits in the digraph  $G$  are *oriented*, i.e. every circuit is given an arbitrary *direction* for the sake of defining the circuit matrix. After choosing the orientations, the circuit matrix of  $G$  is  $\mathbf{B} = (b_{ij})$  where

$$b_{ij} = \begin{cases} 1 & \text{if the arc } e_j \text{ is in the circuit } C_i \text{ and they in the same direction} \\ -1 & \text{if the arc } e_j \text{ is in the circuit } C_i \text{ and they are in the opposite direction} \\ 0 & \text{otherwise.} \end{cases}$$

**Example.** For the directed graph



the circuits are



and the circuit matrix is

$$\mathbf{B} = \begin{pmatrix} e_1 & e_2 & e_3 & e_4 \\ 1 & 0 & -1 & 1 \\ -1 & 1 & 0 & -1 \\ 0 & -1 & 1 & 0 \end{pmatrix} \begin{matrix} C_1 \\ C_2 \\ C_3 \end{matrix}$$

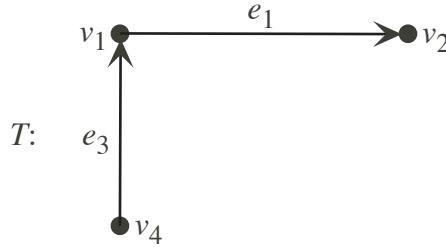
If the graph  $G$  is connected and contains at least one circuit, then it has a cospanning tree  $T^*$  and the corresponding fundamental circuits. By choosing the corresponding rows of the circuit matrix  $\mathbf{B}$ , we get an  $(m - n + 1) \times m$  matrix  $\mathbf{B}_f$ , called the *fundamental circuit matrix*. Similarly, a connected digraph  $G$  with at least one circuit has a fundamental circuit matrix: the direction of a fundamental circuit is the same as the direction of the corresponding link in  $T^*$ .

When we rearrange the edges of  $G$  so that the links of  $T^*$  come last and sort the fundamental circuits in the same order, the fundamental circuit matrix takes the form

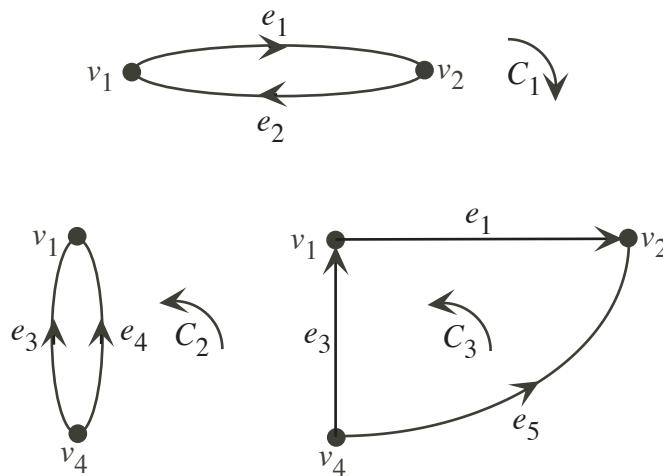
$$\mathbf{B}_f = (\mathbf{B}_{ft} \mid \mathbf{I}_{m-n+1}),$$

where  $\mathbf{I}_{m-n+1}$  is the identity matrix with  $m - n + 1$  rows. The rank of  $\mathbf{B}_f$  is thus  $m - n + 1 = \mu(G)$  and the rank of  $\mathbf{B}$  is  $\geq m - n + 1$ .

**Example.** (Continuing from the previous example) We left out vertex  $v_3$  so we get a connected digraph (see p.34) and we chose the spanning tree



The fundamental circuits are



and

$$\mathbf{B}_f = \left( \begin{array}{ccccc|c} e_1 & e_3 & e_2 & e_4 & e_5 & \\ 1 & 0 & 1 & 0 & 0 & C_1 \\ 0 & -1 & 0 & 1 & 0 & C_2 \\ -1 & -1 & 0 & 0 & 1 & C_3 \end{array} \right)$$

**Theorem 4.3.** *An oriented cut and an oriented circuit of a digraph have an even number of common arcs. Half of these arcs have the same direction in the cut and in the circuit, and the remaining arcs have opposite directions in the cut and in the circuit.*

*Proof.* Compare to the proof of Theorem 2.6.  $\square$

**Theorem 4.4.** *For a digraph,  $BQ^T = O$  (zero matrix).*

*Proof.* By the previous theorem, half of the nonzero numbers in the dot product corresponding to each element of  $BQ^T$  are  $+1$ . The remaining nonzero numbers are  $-1$ . Therefore, the dot product is  $= 0$ .  $\square$

**Theorem 4.5.** *If the digraph  $G$  contains at least one circuit, then the rank of its circuit matrix  $B$  is  $\mu(G)$ . Furthermore, if  $G$  is connected, then the circuit matrix  $B$  can be expressed as  $B = B_2B_f$ , where the matrix  $B_2$  consists of 0's and  $\pm 1$ 's, and the cut matrix  $Q$  can be expressed as  $Q = Q_1Q_f$ , where the matrix  $Q_1$  consists of 0's and  $\pm 1$ 's.*

*Proof.* First we consider the case when  $G$  is connected. We choose a spanning tree  $T$  of  $G$  and rearrange the  $m$  edges of  $G$  so that the branches of  $T$  come first and the links of  $T^*$  come last. We sort the fundamental cut sets in the same order as the branches and links. Then,

$$Q_f = \left( \begin{array}{c|c} I_{n-1} & Q_{fc} \end{array} \right) \quad \text{and} \quad B_f = \left( \begin{array}{c|c} B_{ft} & I_{m-n+1} \end{array} \right).$$

The blocks of  $B$  can be constructed in a similar way:

$$B = \left( \begin{array}{c|c} B_1 & B_2 \end{array} \right).$$

Since  $Q_f$  is a submatrix of  $Q$  and  $B_f$  is a submatrix of  $B$ , it follows from Theorem 4.4 that

$$\begin{aligned} O &= B_f Q_f^T = \left( \begin{array}{c|c} B_{ft} & I_{m-n+1} \end{array} \right) \left( \begin{array}{c|c} I_{n-1} & Q_{fc} \end{array} \right)^T = \left( \begin{array}{c|c} B_{ft} & I_{m-n+1} \end{array} \right) \left( \frac{I_{n-1}}{Q_{fc}^T} \right) \\ &= B_{ft} I_{n-1} + I_{m-n+1} Q_{fc}^T = B_{ft} + Q_{fc}^T. \end{aligned}$$

Hence

$$B_{ft} = -Q_{fc}^T.$$

Furthermore, since  $Q_f$  is a submatrix of  $Q$ , we can use the same theorem to get

$$\begin{aligned} O &= B Q_f^T = \left( \begin{array}{c|c} B_1 & B_2 \end{array} \right) \left( \begin{array}{c|c} I_{n-1} & Q_{fc} \end{array} \right)^T = \left( \begin{array}{c|c} B_1 & B_2 \end{array} \right) \left( \frac{I_{n-1}}{Q_{fc}^T} \right) \\ &= B_1 I_{n-1} + B_2 Q_{fc}^T = B_1 - B_2 B_{ft}. \end{aligned}$$

Hence

$$B = \left( \begin{array}{c|c} B_2 B_{ft} & B_2 \end{array} \right) = B_2 \left( \begin{array}{c|c} B_{ft} & I_{m-n+1} \end{array} \right) = B_2 B_f,$$

as claimed. In the same way,  $Q$  can be expressed as  $Q = Q_1 Q_f$ , as claimed, which is clear anyway since the rank of  $Q$  is  $n - 1$  and its elements are 0's and  $\pm 1$ 's.

Every row of  $B$  is a linear combination of the rows corresponding to the fundamental circuits and the rank of  $B$  is at most equal to the rank of  $B_f = m - n + 1$ . On the other hand, as we pointed out earlier, the rank of  $B$  is  $\geq m - n + 1$ . Thus,  $\text{rank}(B) = m - n + 1 (= \mu(G))$  for a connected digraph.

In the case of a disconnected digraph  $G$  (which contains at least one circuit), it is divided into components ( $k \geq 2$  components) and the circuit matrix  $B$  is divided into blocks corresponding to the components (compare to the proof of the corollary of Theorem 4.1), in which case

$$\text{rank}(B) = \sum_{i=1}^k (m_i - n_i + 1) = m - n + k = \mu(G). \quad \square$$

Notice that the proof also gives the formula,  $\mathbf{B}_{ft} = -\mathbf{Q}_{fc}^T$ , which connects the fundamental cut matrix and the fundamental circuit matrix.

## 4.4 An Application: Stationary Linear Networks

A *stationary linear network* is a directed graph  $G$  that satisfies the following conditions:

1.  $G$  is connected.
2. Every arc of  $G$  belongs to some circuit and there are no loops in  $G$ .
3. Every arc  $e_j$  in  $G$  is associated with a number  $i_j$  called the *through-quantity* or *flow*. If there are  $m$  arcs in  $G$ , then we write

$$\mathbf{i} = \begin{pmatrix} i_1 \\ \vdots \\ i_m \end{pmatrix}$$

(*through-vector*).

4. Every vertex  $v_i$  in  $G$  is associated with a number  $p_i$  called the *potential*. Furthermore, the *across-quantity* or *potential difference* of the arc  $e_j = (v_{i_1}, v_{i_2})$  is

$$u_j = p_{i_2} - p_{i_1}.$$

If there are  $n$  vertices and  $m$  arcs in  $G$ , then we write

$$\mathbf{p} = \begin{pmatrix} p_1 \\ \vdots \\ p_n \end{pmatrix} \quad \text{and} \quad \mathbf{u} = \begin{pmatrix} u_1 \\ \vdots \\ u_m \end{pmatrix}$$

(*potential vector* and *across-vector*). (Potentials are rarely needed.)

5. Every arc  $e_j$  is one of the following:
  - (a) a *component*<sup>1</sup>, for which there is an associated number  $r_j$ .  $r_j$  is constant ( $\neq 0$ ) (stationarity) and the following equation links the quantities:
 
$$u_j = i_j r_j \quad (\text{linearity}).$$
  - (b) a *through-source*, for which the through-quantity  $i_j$  is fixed.
  - (c) an *across-source*, for which the across-quantity  $u_j$  is fixed.
6. (*Kirchhoff's Through-Quantity Law*) The sum of the through-quantities of an oriented cut of  $G$  is zero when the cut is interpreted as an edge set and the sign of a through-quantity is changed if the directions of a cut and an arc are different.
7. (*Kirchhoff's Across-Quantity Law*) The sum of the across-quantities of an oriented circuit of  $G$  is zero when the sign of an across-quantity is changed if the directions of a circuit and an arc are different.

---

<sup>1</sup>Not to be confused with a component of a graph!

**Example.** A typical stationary linear network is an electrical circuit with linear resistors, constant current sources and constant voltage sources. The components are resistors and  $r_j$  are the resistances. Equation 5.(a) is Ohm's Law.

We take a spanning tree  $T$  of a stationary linear network  $G$ , its fundamental cut matrix  $\mathbf{Q}_f$  and its fundamental circuit matrix  $\mathbf{B}_f$ . Let us rearrange the arcs in these matrices and vectors  $\mathbf{i}$  and  $\mathbf{u}$  like we did before. That is, the branches of  $T$  will come first followed by the links of  $T^*$ . Kirchhoff's Laws can then be written as

$$\mathbf{Q}\mathbf{i} = \mathbf{0} \quad \text{and} \quad \mathbf{B}\mathbf{u} = \mathbf{0}.$$

On the other hand, the rows of the fundamental cut matrix  $\mathbf{Q}_f$  span all the rows of  $\mathbf{Q}$ , and similarly rows of the fundamental circuit matrix  $\mathbf{B}_f$  span the rows of  $\mathbf{B}$ . Then, Kirchhoff's Laws can also be written as

$$\mathbf{Q}_f\mathbf{i} = \mathbf{0}_{n-1} \quad \text{and} \quad \mathbf{B}_f\mathbf{u} = \mathbf{0}_{m-n+1}.$$

Let us form the diagonal matrices  $\mathbf{K} = [k_1, \dots, k_m]$  and  $\mathbf{L} = [\ell_1, \dots, \ell_m]$ , where

$$k_j = \begin{cases} -r_j & \text{if } e_j \text{ is a component} \\ 1 & \text{if } e_j \text{ is a through-source} \\ 0 & \text{if } e_j \text{ is an across-source} \end{cases} \quad \text{and} \quad \ell_j = \begin{cases} 1 & \text{if } e_j \text{ is a component} \\ 0 & \text{if } e_j \text{ is a through-source} \\ 1 & \text{if } e_j \text{ is an across-source,} \end{cases}$$

and the  $m$ -vector  $\mathbf{s} = (s_1, \dots, s_m)^T$ , where

$$s_j = \begin{cases} 0 & \text{if } e_j \text{ is a component} \\ i_j & \text{if } e_j \text{ is a through-source} \\ u_j & \text{if } e_j \text{ is an across-source.} \end{cases}$$

Then, all the information can be expressed as a system of linear equations

$$\left( \begin{array}{c|c} \mathbf{K} & \mathbf{L} \\ \hline \mathbf{Q}_f & \mathbf{0}_{(n-1) \times m} \\ \hline \mathbf{0}_{(m-n+1) \times m} & \mathbf{B}_f \end{array} \right) \left( \begin{array}{c} \mathbf{i} \\ \mathbf{u} \end{array} \right) = \left( \begin{array}{c} \mathbf{s} \\ \mathbf{0}_{n-1} \\ \mathbf{0}_{m-n+1} \end{array} \right),$$

known as the *fundamental equations*. The through and across quantities can be solved (ideally) if  $r_j$  and the sources are given.

**Remark.** The same procedure can be applied to form state (differential) equations for dynamic networks, which have nonstationary components.

The matrix of this system of linear equations does not have to be nonsingular and the system does not even have to have a unique solution at all. For example, in the matrix above, we can easily see that it is singular if some circuit only consists of across-sources or if some cut only consists of through-sources. As a matter of fact, this is the only case when the through and across quantities are not defined uniquely if the constants  $r_j$  are real numbers with the same sign (and often otherwise too).

We choose a specific spanning tree  $T$  to explore these concepts more carefully:

**Lemma.** If no cut of  $G$  consists of only through-sources and no circuit of  $G$  consists of only across-sources, then  $G$  has a spanning tree  $T$  such that every across-source is a branch of  $T$  and every through-source is a link of  $T^*$ .

*Proof.* If  $G$  satisfies the hypothesis, then we first choose a digraph  $M$  which has every vertex and across-source (arc) of  $G$ . There are no circuits in this digraph. Then we add components to  $M$  one by one and try to come up with a spanning tree. If this fails at some point, then  $G$  has a cut with only through-sources, which is impossible.  $\square$

Now let us assume that no cut of  $G$  consists of only through-sources and no circuit of  $G$  consists of only across-sources. We use the spanning tree  $T$  mentioned in the lemma. We rearrange the arcs of  $G$  so that (as before) the branches of  $T$  come first. Within these branches, the across-sources come first followed by components. Similarly, the links are rearranged so that the components come first and the through-sources come last.

The system of  $2m$  equations can then be written as

$$\begin{pmatrix} \mathbf{O} & \mathbf{O} & \mathbf{O} & \mathbf{O} & \mathbf{I} & \mathbf{O} & \mathbf{O} & \mathbf{O} \\ \mathbf{O} & -\mathbf{R}_1 & \mathbf{O} & \mathbf{O} & \mathbf{O} & \mathbf{I} & \mathbf{O} & \mathbf{O} \\ \mathbf{O} & \mathbf{O} & -\mathbf{R}_2 & \mathbf{O} & \mathbf{O} & \mathbf{O} & \mathbf{I} & \mathbf{O} \\ \mathbf{O} & \mathbf{O} & \mathbf{O} & \mathbf{I} & \mathbf{O} & \mathbf{O} & \mathbf{O} & \mathbf{O} \\ \mathbf{I} & \mathbf{O} & \mathbf{Q}_{11} & \mathbf{Q}_{12} & \mathbf{O} & \mathbf{O} & \mathbf{O} & \mathbf{O} \\ \mathbf{O} & \mathbf{I} & \mathbf{Q}_{21} & \mathbf{Q}_{22} & \mathbf{O} & \mathbf{O} & \mathbf{O} & \mathbf{O} \\ \mathbf{O} & \mathbf{O} & \mathbf{O} & \mathbf{O} & \mathbf{B}_{11} & \mathbf{B}_{12} & \mathbf{I} & \mathbf{O} \\ \mathbf{O} & \mathbf{O} & \mathbf{O} & \mathbf{O} & \mathbf{B}_{21} & \mathbf{B}_{22} & \mathbf{O} & \mathbf{I} \end{pmatrix} \begin{pmatrix} \mathbf{i}_1 \\ \mathbf{i}_2 \\ \mathbf{i}_3 \\ \mathbf{i}_4 \\ \mathbf{u}_1 \\ \mathbf{u}_2 \\ \mathbf{u}_3 \\ \mathbf{u}_4 \end{pmatrix}$$

$$= \begin{pmatrix} \mathbf{s}_1 \\ \mathbf{0} \\ \mathbf{0} \\ \mathbf{s}_2 \\ \mathbf{0} \\ \mathbf{0} \\ \mathbf{0} \\ \mathbf{0} \end{pmatrix} \begin{array}{l} \leftarrow \text{across-sources (in branches)} \\ \leftarrow \text{components (in branches)} \\ \leftarrow \text{components (in links)} \\ \leftarrow \text{through-sources (in links)} \\ \leftarrow \text{fundamental cut sets (across-sources)} \\ \leftarrow \text{fundamental cut sets (components)} \\ \leftarrow \text{fundamental circuits (components)} \\ \leftarrow \text{fundamental circuits (through-sources)} \end{array}$$

where the  $\mathbf{I}$ 's are identity matrices of the right dimensions, the  $\mathbf{O}$ 's are zero matrices of the right dimensions and the  $\mathbf{0}$ 's are zero vectors of the right dimensions.

**Remark.** Here we assume that  $G$  has all these four types of arcs (across-source branch, component branch, through-source link and component link). In other cases (for example, when there are no through-sources), we leave the corresponding rows, columns and elements out of the system of equations. Other cases are treated in a similar way.

Solving the equations, we get

$$\mathbf{u}_1 = \mathbf{s}_1 , \quad \mathbf{u}_2 = \mathbf{R}_1 \mathbf{i}_2 , \quad \mathbf{u}_3 = \mathbf{R}_2 \mathbf{i}_3 , \quad \mathbf{i}_4 = \mathbf{s}_2$$

which leaves this system of equations:

$$\begin{pmatrix} \mathbf{I} & \mathbf{O} & \mathbf{Q}_{11} & \mathbf{O} \\ \mathbf{O} & \mathbf{I} & \mathbf{Q}_{21} & \mathbf{O} \\ \mathbf{O} & \mathbf{B}_{12}\mathbf{R}_1 & \mathbf{R}_2 & \mathbf{O} \\ \mathbf{O} & \mathbf{B}_{22}\mathbf{R}_1 & \mathbf{O} & \mathbf{I} \end{pmatrix} \begin{pmatrix} \mathbf{i}_1 \\ \mathbf{i}_2 \\ \mathbf{i}_3 \\ \mathbf{u}_4 \end{pmatrix} = - \begin{pmatrix} \mathbf{Q}_{12}\mathbf{s}_2 \\ \mathbf{Q}_{22}\mathbf{s}_2 \\ \mathbf{B}_{11}\mathbf{s}_1 \\ \mathbf{B}_{21}\mathbf{s}_1 \end{pmatrix}.$$

Thus,

$$\mathbf{i}_1 = -\mathbf{Q}_{11}\mathbf{i}_3 - \mathbf{Q}_{12}\mathbf{s}_2 , \quad \mathbf{i}_2 = -\mathbf{Q}_{21}\mathbf{i}_3 - \mathbf{Q}_{22}\mathbf{s}_2 , \quad \mathbf{u}_4 = -\mathbf{B}_{22}\mathbf{R}_1\mathbf{i}_2 - \mathbf{B}_{21}\mathbf{s}_1.$$

From the results in the previous section, we get

$$\mathbf{B}_{ft} = \left( \begin{array}{c|c} \mathbf{B}_{11} & \mathbf{B}_{12} \\ \hline \mathbf{B}_{21} & \mathbf{B}_{22} \end{array} \right) = -\mathbf{Q}_{fc}^T = -\left( \begin{array}{c|c} \mathbf{Q}_{11} & \mathbf{Q}_{12} \\ \hline \mathbf{Q}_{21} & \mathbf{Q}_{22} \end{array} \right)^T = \left( \begin{array}{c|c} -\mathbf{Q}_{11}^T & -\mathbf{Q}_{21}^T \\ \hline -\mathbf{Q}_{12}^T & -\mathbf{Q}_{22}^T \end{array} \right).$$

Therefore,  $\mathbf{B}_{11} = -\mathbf{Q}_{11}^T$  and  $\mathbf{B}_{12} = -\mathbf{Q}_{21}^T$ . Finally, there is only one set of equations for  $\mathbf{i}_3$ :

$$(\mathbf{Q}_{21}^T \mathbf{R}_1 \mathbf{Q}_{21} + \mathbf{R}_2) \mathbf{i}_3 = \mathbf{Q}_{11}^T \mathbf{s}_1 - \mathbf{Q}_{21}^T \mathbf{R}_1 \mathbf{Q}_{22} \mathbf{s}_2.$$

The matrix<sup>2</sup> of this system of equations can be written as

$$\mathbf{Q}_{21}^T \mathbf{R}_1 \mathbf{Q}_{21} + \mathbf{R}_2 = (\mathbf{Q}_{21}^T | \mathbf{I}) \left( \begin{array}{c|c} \mathbf{R}_1 & \mathbf{O} \\ \hline \mathbf{O} & \mathbf{R}_2 \end{array} \right) \left( \begin{array}{c} \mathbf{Q}_{21} \\ \hline \mathbf{I} \end{array} \right).$$

We can see that it is not singular if the diagonal elements of  $\mathbf{R}_1$  and  $\mathbf{R}_2$  are all positive or all negative.

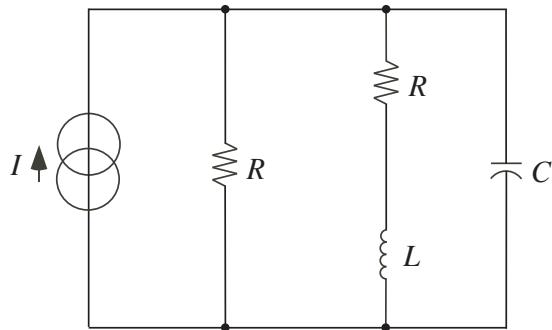
Therefore, we get

**Theorem 4.6.** *If the constants  $r_j$  are real numbers with the same sign, then the fundamental equations of the stationary linear network have a unique solution exactly when no cut of the network consists of only through-sources and no circuit of the network consists of only across-sources.*

From the theorem above, we notice that the number of equations we have to solve (numerically) is considerably fewer than  $2m$ .

**Example.** A mono-frequency AC circuit with passive elements (resistors, capacitors and inductors) can also be modelled as a stationary linear network (Theorem 4.6 does not apply). In the circuit below, the component values are  $R = 10 \Omega$ ,  $C = 100 \mu\text{F}$ ,  $L = 10 \text{ mH}$  and the current source is

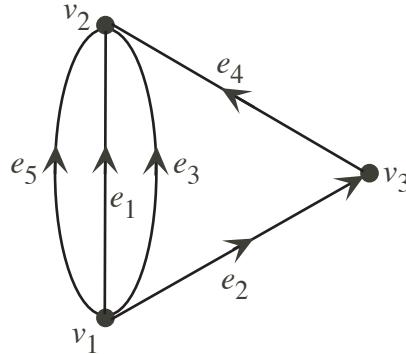
$$I = 10 \cos(1000t) \text{ A.}$$



The complex current of the source is  $10e^{j1000t}$ , where  $j$  is the imaginary unit. The (angular) frequency is  $\omega = 1000 \text{ rad/s}$ . There are no voltage sources. The corresponding digraph is

---

<sup>2</sup>This matrix is called the *impedance matrix*. Similarly, the *admittance matrix* can be constructed from the blocks of the fundamental circuit matrix



The voltages and currents written as complex exponentials are  $i_k = I_k e^{j1000t}$  and  $u_k = U_k e^{j1000t}$ . In particular, the current source is  $i_5 = s_5 = 10e^{j1000t}$ . We get  $r_k$  from the familiar formulae from electrical circuit analysis:

$$r_1 = r_4 = R = 10 \quad , \quad r_3 = \frac{1}{j\omega C} = -10j \quad , \quad r_2 = j\omega L = 10j.$$

We choose the arcs  $e_1$  and  $e_2$  as the branches of the spanning tree  $T$ . Because of the linearity of the system of equations, the exponential factors  $e^{j1000t}$  cancel out and we get

$$\left( \begin{array}{cc|ccccc} -10 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & -10j & 0 & 0 & 0 & 1 & 0 & 0 \\ \hline 0 & 0 & 10j & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & -10 & 0 & 0 & 0 & 1 \\ \hline 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & -1 & 0 & 0 & 0 & 0 \\ \hline 0 & 0 & 0 & 0 & 0 & -1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & -1 & 1 & 0 \\ \hline 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 \end{array} \right) \begin{pmatrix} I_1 \\ I_2 \\ \hline I_3 \\ I_4 \\ \hline I_5 \\ \hline U_1 \\ U_2 \\ \hline U_3 \\ U_4 \\ \hline U_5 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ \hline 0 \\ 0 \\ \hline 10 \\ \hline 0 \\ 0 \\ \hline 0 \\ 0 \\ \hline 0 \end{pmatrix}.$$

Notice that we have left out the across-sources because there are none. This system is easily solved using computer programs, e.g. MATLAB:

```

»H=[-10      0      0      0      1      0      0      0      0 ;
   0     -10*j    0      0      0      1      0      0      0 ;
   0      0     10*j    0      0      0      0      1      0 ;
   0      0      0     -10    0      0      0      0      1 ;
   0      0      0      1      0      0      0      0      0 ;
   1      0      1      1      1      0      0      0      0 ;
   0      1      0     -1      0      0      0      0      0 ;
   0      0      0      0     -1      0      1      0      0 ;
   0      0      0      0     -1      1      0      1      0 ;
   0      0      0      0     -1      0      0      0      1];
»s=[0 0 0 0 10 0 0 0 0 0 0 0]';
»UV=inv(H)*s;
»[UV angle(UV) abs(UV)]
```

ans =

-6.0000 + 2.0000i	2.8198	6.3246
-2.0000 + 4.0000i	2.0344	4.4721
-2.0000 - 6.0000i	-1.8925	6.3246
-2.0000 + 4.0000i	2.0344	4.4721
10.0000	0	10.0000
-60.0000 +20.0000i	2.8198	63.2456

-40.0000	-20.0000i	-2.6779	44.7214
-60.0000	+20.0000i	2.8198	63.2456
-20.0000	+40.0000i	2.0344	44.7214
-60.0000	+20.0000i	2.8198	63.2456

Thus, for example, the complex voltage across the current source is

$$u_5 = U_5 e^{j1000t} = 63.25 e^{j(1000t+2.82)}$$

and the real voltage is  $63.25 \cos(1000t + 2.82)$  V.

Kirchhoff's Through-Quantity Law can also be written in the form

$$\mathbf{Ai} = \mathbf{0}_n,$$

where  $\mathbf{A}$  is the all-vertex incidence matrix of  $G$ . Furthermore,

$$\mathbf{A}^T \mathbf{p} = -\mathbf{u}.$$

Hence

$$\mathbf{u} \bullet \mathbf{i} = \mathbf{u}^T \mathbf{i} = -\mathbf{p}^T \mathbf{A} \mathbf{i} = 0.$$

This result only depends on the structure of the digraph  $G$  (through the all-vertex incidence matrix). Now we get the famous theorem:

**Theorem 4.7. (Tellegen)** *If two stationary linear networks have the same digraph with corresponding through-vectors  $\mathbf{i}_1$  and  $\mathbf{i}_2$  as well as corresponding across-vectors  $\mathbf{u}_1$  and  $\mathbf{u}_2$ , then*

$$\mathbf{u}_1 \bullet \mathbf{i}_2 = 0 \quad \text{and} \quad \mathbf{u}_2 \bullet \mathbf{i}_1 = 0.$$

If we apply this to the case when the two networks are exactly the same ( $= G$ ), then we get

$$\mathbf{u} \bullet \mathbf{i} = 0,$$

known as the *Law of Conservation of Energy*.

**Remark.** More details on this subject can be found e.g. in SWAMY & THULASIRAMAN or VÁGÓ, as well as DOLAN & ALDOUS.

## 4.5 Matrices over GF(2) and Vector Spaces of Graphs

The set  $\{0, 1\}$  is called a *field* (i.e. it follows the same arithmetic rules as real numbers) if addition and multiplication are defined as follows:

	+		×	
	0	1	0	1
0	0	1	0	0
1	1	0	1	0

In this case  $-1 = 1$  and  $1^{-1} = 1$ . This is the field GF(2).

If we think of the elements 0 and 1 of the all-vertex incidence, cut, fundamental cut, circuit and fundamental circuit matrices of a ("undirected") graph as elements of the field GF(2), then Theorems 4.1, 4.2, 4.4, 4.5 and their corollaries also apply to "undirected graphs". (Keep in mind that  $-1 = 1$  in the field GF(2).) The proofs are the same.

For "undirected" graphs, the vector spaces are over the field  $\text{GF}(2)$ . For directed graphs, the vector spaces are real (i.e. over the field  $\mathbb{R}$ ). The row space of the cut matrix of a (di)graph is the *cut space*. Similarly, the row space of the circuit matrix is the *circuit space*. The dimension of the cut space is the rank of the (di)graph and the dimension of the circuit space is the nullity of the (di)graph. Furthermore, the cut space and the circuit space are orthogonal complements. (All of these statements follow directly from the results above.)

Often, we deal with the above mentioned spaces through subgraphs, i.e. we identify a vector with the subgraph generated by the corresponding arcs. In the case of "undirected" graphs, the addition of  $\text{GF}(2)$  vectors corresponds to the ring sum operation.

# Chapter 5

## Graph Algorithms

### 5.1 Computational Complexity of Algorithms

The *complexity* of a problem is related to the resources required to compute a solution as a function of the size of the problem. The size of a problem is measured by the size of the input  $N$ , and the resources required are usually measured by time (number of steps) and space (maximum amount of memory measured appropriately). *Decision problems* or *yes-or-no questions* are very common. Read HOPCROFT & ULLMAN for classical complexity theory.

To make computational complexities comparable, we need to agree on some specific mathematical models for algorithms. For example, consider computing with Turing Machines and refer to courses in Theoretical Computer Science and Mathematical Logic. We have *deterministic* and *nondeterministic* version of algorithm models. In the deterministic version, there are no choices to be made. In the nondeterministic version, there is a choice to be made somewhere on the way. For a nondeterministic algorithm, we have to make the following assumptions so that we can actually solve problems:

1. The algorithm terminates at some point no matter how we choose the steps.
2. The algorithm can terminate without yielding a solution.
3. When the algorithm terminates and yields a solution, the solution is correct (it is possible to have more than one solution).
4. For decision problems, if the algorithm fails to give a positive answer (yes), then the answer is interpreted to be negative (no).
5. If the problem is to compute a value, then the nondeterministic algorithm has to give a solution for every input (value of the function).

Nondeterministic algorithms are best treated as *verification procedures* for problems rather than procedures for producing answers.

Computational complexity is considered *asymptotically*, that is for large problems, time or space complexities that differ by constant coefficients are not distinguished because linear acceleration and compression of space are easy to perform in any kind of algorithm model. Although the choice of an algorithm model has a clear impact on the complexity, it is not an essential characteristic, i.e. it does not change the complexity class. Often, we use the *big-O notation* for complexities.  $O(f(N))$  refers to the class of functions  $g(N)$  such that if  $N \geq N_0$  holds, then  $|g(N)| \leq Cf(N)$  holds, where  $C$  is a constant.

Without exploring algorithm models any further, we define a couple of important complexity classes. The time complexity class  $\mathcal{P}$  (*deterministic polynomial time problems*) consists of problems of (input) size  $N$  where it takes at most  $p(N)$  steps to solve the problem using deterministic algorithms.  $p(N)$  is some problem dependent polynomial of  $N$ . The time complexity class  $\mathcal{NP}$  (*nondeterministic polynomial time problems*) consists of problems of size  $N$  where it takes at most  $p(N)$  steps to solve the problem using nondeterministic algorithms. Once again,  $p(N)$  is some problem dependent polynomial of  $N$ .

Time complexity class  $\text{co-}\mathcal{NP}$  (*complements of nondeterministic polynomial time problems*) consists of decision problems whose complements are in  $\mathcal{NP}$ . (The *complement* of a problem is obtained by swapping the positive and the negative answer.)

Obviously,  $\mathcal{P} \subseteq \mathcal{NP}$  and (for decision problems)  $\mathcal{P} \subseteq \text{co-}\mathcal{NP}$ . Whether or not the inclusion is proper is an open problem, actually quite a famous problem. It is widely believed that both of the inclusions are proper. It is not known if the following holds for decision problems:

$$\mathcal{NP} = \text{co-}\mathcal{NP} \quad \text{or} \quad \mathcal{P} = \mathcal{NP} \cap \text{co-}\mathcal{NP}$$

Most researchers believe that they do not hold.

The space complexity class  $\mathcal{PSPACE}$  (*deterministic polynomial space problems*) consists of problems of (input) size  $N$  where it takes at most  $p(N)$  memory units to solve the problem using deterministic algorithms.  $p(N)$  is some problem dependent polynomial of  $N$ . The space complexity class  $\mathcal{NPSPACE}$  (*nondeterministic polynomial space problems*) consists of problems of size  $N$  where it takes at most  $p(N)$  memory units to solve the problem using nondeterministic algorithms. Once again,  $p(N)$  is some problem dependent polynomial of  $N$ . It is known that

$$\mathcal{NP} \subseteq \mathcal{PSPACE} = \mathcal{NPSPACE},$$

but it is not known whether the inclusion is proper or not.

An algorithm may include some ideally generated random numbers. The algorithm is then called *probabilistic* or *stochastic*. The corresponding polynomial time complexity class is  $\mathcal{BPP}$  (*random polynomial time problems* or *bounded-error probabilistic polynomial time problems*). Some stochastic algorithms may fail occasionally, that is, they produce no results and terminate prematurely. These algorithms are called *Las Vegas algorithms*. Some stochastic algorithms may also produce wrong answers (ideally with a small probability). These kind of algorithms are called *Monte Carlo algorithms*. Some stochastic algorithms seldom yield exact solutions. Nevertheless, they give accurate approximate solutions with high probability. These kind of algorithms are called *approximation algorithms*.

The task of an algorithm may be to convert a problem to another. This is known as *reduction*. If problem  $A$  can be reduced to another problem  $B$  by using a (deterministic) polynomial time algorithm, then we can get a polynomial time algorithm for problem  $A$  from a polynomial time algorithm for  $B$ . A problem is  $\mathcal{NP}$ -hard if every problem in  $\mathcal{NP}$  can be reduced to it by a polynomial time algorithm.  $\mathcal{NP}$ -hard problems are  $\mathcal{NP}$ -complete if they are actually in  $\mathcal{NP}$ .  $\mathcal{NP}$ -complete problems are the "worst kind". If any problem in  $\mathcal{NP}$  could be shown to be deterministic polynomial time, then every problem in  $\mathcal{NP}$  would be in  $\mathcal{P}$  and  $\mathcal{P} = \mathcal{NP}$ . Over one thousand  $\mathcal{NP}$ -complete problems are known currently.

The old division of problems into *tractable* and *intractable* means that  $\mathcal{P}$  problems are tractable and others are not. Because we believe that  $\mathcal{P} \neq \mathcal{NP}$  in general,  $\mathcal{NP}$ -complete problems are intractable. In the following, graph algorithms are either in  $\mathcal{P}$  or they are approximations of some more demanding problems. The size of an input can be for example the number of nonzero elements in an incidence matrix, the number of vertices  $n$  or the number of edges  $m$  or some combination of  $n$  and  $m$ .

## 5.2 Reachability: Warshall's Algorithm

We only deal with directed graphs in this section. The results also hold for "undirected" graphs if we interpret an edge as a pair of arcs in opposite directions.

**Problem.** We are given an adjacency matrix of the digraph  $G = (V, E)$ . We are to construct the reachability matrix  $\mathbf{R} = (r_{ij})$  of  $G$ , where

$$r_{ij} = \begin{cases} 1 & \text{if } G \text{ has a directed } v_i-v_j \text{ path} \\ 0 & \text{otherwise.} \end{cases}$$

(Note that  $V = \{v_1, \dots, v_n\}$ .) In particular, we should note that if  $r_{ii} = 1$ , then  $v_i$  is in a directed circuit.

Warshall's Algorithm constructs a series of  $n \times n$  matrices  $\mathbf{E}_1, \dots, \mathbf{E}_n$  where

1. elements of  $\mathbf{E}_i$  are either zero or one.
2.  $\mathbf{E}_i \leq \mathbf{E}_{i+1}$  ( $i = 0, \dots, n - 1$ ) (comparison is done element by element).
3.  $\mathbf{E}_0$  is obtained from the adjacency matrix  $\mathbf{D}$  by replacing the positive elements with ones.
4.  $\mathbf{E}_n = \mathbf{R}$ .

The algorithm is presented as a pseudocode:

```

procedure Warshall
begin
     $\mathbf{E} := \mathbf{E}_0$ 
    for  $i := 1$  to  $n$  do
        for  $j := 1$  to  $n$  do
            if  $(\mathbf{E})_{ji} = 1$  then for  $k := 1$  to  $n$  do
                 $(\mathbf{E})_{jk} := \max((\mathbf{E})_{jk}, (\mathbf{E})_{ik})$ 
            fi
        od
    od
end
```

In this case, the maximizing operation is sometimes called the *Boolean sum*:

$$\begin{array}{c|cc} \max & 0 & 1 \\ \hline 0 & 0 & 1 \\ 1 & 1 & 1 \end{array}$$

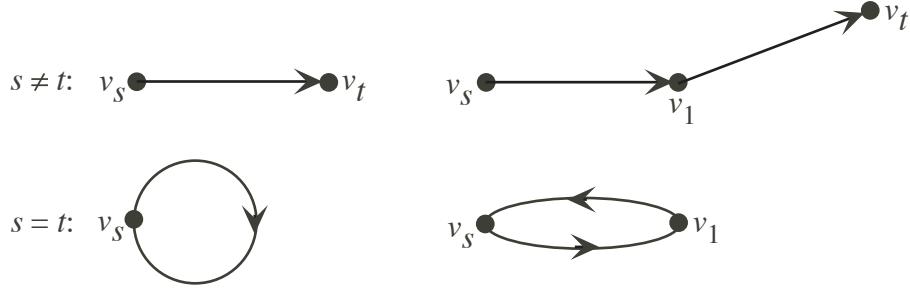
Let us show that Warshall's Algorithm gives us the desired results. Let  $\mathbf{E}_i$  denote the value of  $\mathbf{E}$  after  $i$  steps.

**Statement.** (i) If there is a directed path from  $v_s$  to  $v_t$  such that apart from  $v_s$  and  $v_t$ , the path only includes vertices in the set  $\{v_1, \dots, v_i\}$ , then  $(\mathbf{E}_i)_{st} = 1$ .

(ii) If vertex  $v_s$  belongs to a directed circuit whose other vertices are in the set  $\{v_1, \dots, v_i\}$ , then  $(\mathbf{E}_i)_{ss} = 1$ .

*Proof.* We will use induction on  $i$ .

Induction Basis:  $i = 1$ .  $(\mathbf{E}_1)_{st} = 1$  if  $(\mathbf{E}_0)_{st} = 1$ , or  $(\mathbf{E}_0)_{s1} = 1$  and  $(\mathbf{E}_0)_{1t} = 1$ . We have one of the following cases:



Induction Hypothesis: The statement is true for  $i < \ell$ . ( $\ell \geq 2$ )

Induction Statement: The statement is true for  $i = \ell$ .

Induction Statement Proof: Let us handle both statements together. The proof for (ii) is given in square brackets. We have two cases:

- $v_\ell$  belongs to the directed path [resp. directed circuit] but  $\ell \neq s, t$  [resp.  $\ell \neq s$ ]. Then, we use the Induction Hypothesis:

$$(\mathbf{E}_{\ell-1})_{s\ell} = 1 \quad \text{and} \quad (\mathbf{E}_{\ell-1})_{\ell t} = 1 \quad [\text{resp. } (\mathbf{E}_{\ell-1})_{s\ell} = 1 \quad \text{and} \quad (\mathbf{E}_{\ell-1})_{\ell s} = 1],$$

$$\text{so } (\mathbf{E}_\ell)_{st} = 1 \quad [\text{resp. } (\mathbf{E}_\ell)_{ss} = 1].$$

- $v_\ell$  is either  $v_s$  or  $v_t$  [resp.  $v_\ell$  is  $v_s$ ] or it does not belong to the directed path [resp. directed circuit] at all. Then, by the Induction Hypothesis

$$(\mathbf{E}_{\ell-1})_{st} = 1 \quad [\text{resp. } (\mathbf{E}_{\ell-1})_{ss} = 1],$$

$$\text{so } (\mathbf{E}_\ell)_{st} = 1 \quad [\text{resp. } (\mathbf{E}_\ell)_{ss} = 1]. \quad \square$$

In Warshall's Algorithm, the maximizing operation is performed at most  $n^3$  times.

### 5.3 Depth-First and Breadth-First Searches

**Problem.** We have to traverse through a (di)graph to find some kind of vertices or edges.

We assume that the (di)graph is connected and loopless. For disconnected graphs, we have to go through the components separately. We ignore loops if the (di)graph has any.

*Depth-First Search, DFS*, has many uses. The procedure is a bit different for undirected graphs and directed graphs. Therefore they will be treated separately.

#### Undirected Graphs

We choose a starting vertex  $r$  (*root*) to start the search. Then, we traverse an edge  $e = (r, v)$  to go to the vertex  $v$ . At the same time, we *direct*  $e$  from  $r$  to  $v$ . Now, we say that the edge  $e$  is *examined* and we call it a *tree edge*. The vertex  $r$  is called the *father* of  $v$  and we denote it  $r = \text{FATHER}(v)$ .

We continue the search. At a vertex  $x$ , there are two cases:

- (1) If every edge incident to  $x$  has been examined, return to the father of  $x$  and continue the process from  $\text{FATHER}(x)$ . The vertex  $x$  is said to be *completely scanned*.
- (2) If there exist some unexamined edges incident to  $x$ , then we choose one such edge  $e = (x, y)$  and direct it from  $x$  to  $y$ . This edge is now said to be *examined*. We have two subcases now:
  - (2.1) If  $y$  has not been visited before, then we traverse the edge  $(x, y)$ , visit  $y$  and continue the search from  $y$ . In this case,  $e$  is a *tree edge* and  $\text{FATHER}(y) = x$ .
  - (2.2) If  $y$  has been visited before, then we select some other unexamined edge incident to  $x$ . In this case, the edge  $e$  is called a *back edge*.

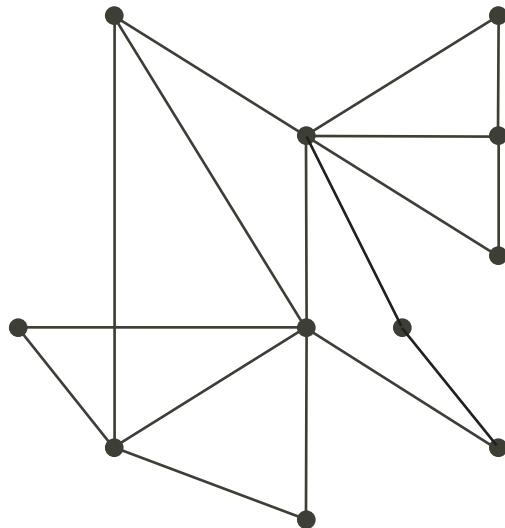
Every time we come to a new vertex which has never been visited before, we give it a distinct number. The number of the root is 1. We write

$$\text{DFN}(x) = \text{running number of vertex } x.$$

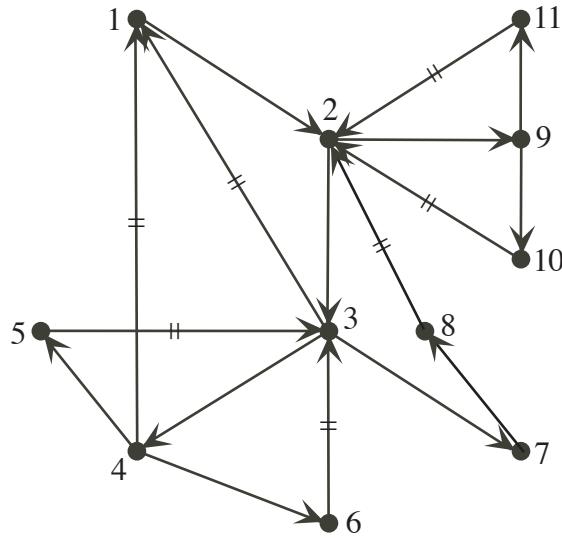
A complete DFS ends when we traverse back to the root and we have visited every vertex or when we have found the desired edge/vertex.

DFS divides the edges of  $G$  into tree edges and back edges. Obviously, the tree edges form a spanning tree of  $G$ , also known as a *DFS tree*. If we include the directions of the tree edges, we get a *directed DFS tree*. DFS gives a direction to every edge in  $G$ . When we use these directions, we get a digraph whose underlying subgraph is  $G$ . It has the DFS tree as a directed spanning tree.

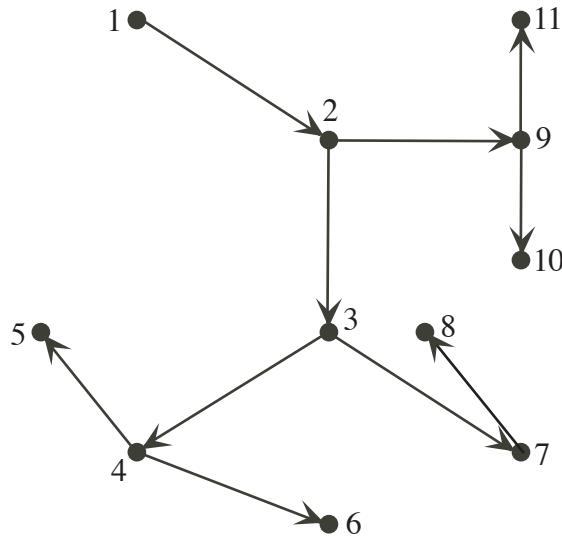
**Example.** For the graph



we start the DFS from a root in the upper left corner. The back edges are marked with two lines.



The corresponding DFS tree is



In the following, we denote,

$$K(x) = \begin{cases} 0 & \text{if vertex } x \text{ has not been visited} \\ 1 & \text{if vertex } x \text{ has been visited} \end{cases}$$

and TREE and BACK are set variables containing the directed tree edges and back edges.

### Depth-First Search for Graphs:

1. Set TREE  $\leftarrow \emptyset$ , BACK  $\leftarrow \emptyset$  and  $i \leftarrow 1$ . For every vertex  $x$  of  $G$ , set FATHER( $x$ )  $\leftarrow 0$  and  $K(x) \leftarrow 0$ .
2. Choose a vertex  $r$  for which  $K(r) = 0$  (this condition is needed only for disconnected graphs, see step #6). Set DFN( $r$ )  $\leftarrow i$ ,  $K(r) \leftarrow 1$  and  $u \leftarrow r$ .
3. If every edge incident to  $u$  has been examined, go to step #5. Otherwise, choose an edge  $e = (u, v)$  that has not been examined.

4. We direct edge  $e$  from  $u$  to  $v$  and label it examined.
- 4.1 If  $K(v) = 0$ , then set  $i \leftarrow i + 1$ ,  $\text{DFN}(v) \leftarrow i$ ,  $\text{TREE} \leftarrow \text{TREE} \cup \{e\}$ ,  $K(v) \leftarrow 1$ ,  $\text{FATHER}(v) \leftarrow u$  and  $u \leftarrow v$ . Go back to step #3.
- 4.2 If  $K(v) = 1$ , then set  $\text{BACK} \leftarrow \text{BACK} \cup \{e\}$  and go back to step #3.
5. If  $\text{FATHER}(u) \neq 0$ , then set  $u \leftarrow \text{FATHER}(u)$  and go back to step #3.
6. (Only for disconnected graphs so that we can jump from one component to another.) If there is a vertex  $r$  such that  $K(r) = 0$ , then set  $i \leftarrow i + 1$  and go back to step #2.
7. Stop.

We denote  $T$  as the DFS tree and  $\vec{G}$  as the directed graph obtained from the algorithm.  $T$  is a directed spanning tree of  $\vec{G}$ . If there is a directed path from  $u$  to  $v$  in  $T$ , then we call  $u$  an *ancestor* of  $v$  and  $v$  a *descendent* of  $u$ . Vertices  $u$  and  $v$  are *related* if one of them is an ancestor of the other. In particular, if  $(u, v)$  is an edge of  $T$ , then  $u$  is the father of  $v$  and  $v$  is a *son* of  $u$ . An edge  $(u, v)$  of  $G$ , where  $u$  and  $v$  are unrelated, is called a *cross edge*. However,

**Statement.** *Cross edges do not exist.*

*Proof.* Let  $u$  and  $v$  be two distinct vertices which are unrelated. Then, (by quasi-strong connectivity) there are two vertices  $u'$  and  $v'$  such that

- $\text{FATHER}(u') = \text{FATHER}(v')$ ,
- $u' = u$  or  $u'$  is an ancestor of  $u$  and
- $v' = v$  or  $v'$  is an ancestor of  $v$ .

We examine the case where  $\text{DFN}(u') < \text{DFN}(v')$  (the other case is obviously symmetrical). We label  $T_1$  as the directed subtree of  $T$  whose root is  $u'$  and  $T_2$  as the directed subtree of  $T$  whose root is  $v'$ . Obviously, DFS goes through the vertices of  $T_2$  only after  $u'$  is completely scanned. Furthermore,  $u'$  is completely scanned only after all the vertices of  $T_1$  are completely scanned. Hence, it is impossible to have an edge  $(u, v)$ .  $\square$

## Directed Graphs

Depth-first search in a (connected and loopless) digraph  $G$  is similar to the case for undirected graphs. The algorithm divides the arcs in  $G$  into four different classes. If the search proceeds to an unexamined arc  $e = (x, y)$ , then the four possible classes are:

- (1) If  $y$  has not been visited, then  $e$  is a *tree edge*.
- (2) If  $y$  has been visited, then there are three cases:
  - (2.1)  $y$  is a descendent of  $x$  in the subgraph induced by existing tree edges. Then,  $e$  is a *forward edge* and  $\text{DFN}(y) > \text{DFN}(x)$ .
  - (2.2)  $x$  is a descendent of  $y$  in the subgraph induced by the existing tree edges. Then,  $e$  is a *back edge* and  $\text{DFN}(y) < \text{DFN}(x)$ .
  - (2.3)  $x$  and  $y$  are not related by any of the existing tree edges. Then,  $e$  is a *cross edge* and  $\text{DFN}(y) < \text{DFN}(x)$ . (*Note!* It is impossible that  $\text{DFN}(y) > \text{DFN}(x)$ . This is proven in the same way as we did previously.)

The directed subgraph of  $G$  induced by tree edges is called the *DFS forest* (directed forest).

If  $\text{DFN}(y) > \text{DFN}(x)$  holds for the arc  $(x, y)$ , then  $(x, y)$  is a tree edge or a forward edge. During the search, it is easy to distinguish the two because  $(x, y)$  is a tree edge if  $y$  has not been visited and it is a forward edge otherwise. If  $\text{DFN}(y) < \text{DFN}(x)$ , then  $(x, y)$  is a back edge or a cross edge. During the search, it is easy to distinguish the two because  $(x, y)$  is a cross edge if  $y$  is completely scanned and it is a back edge otherwise.

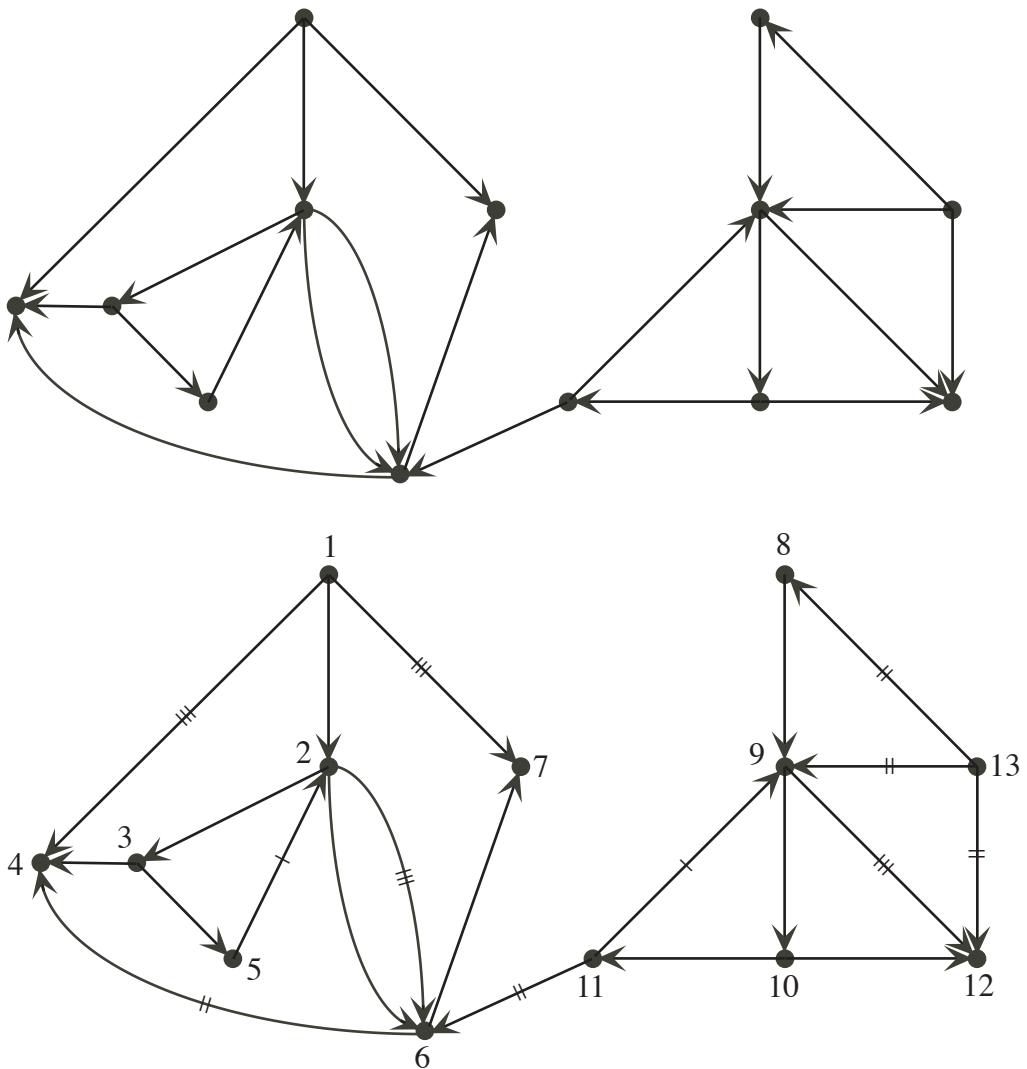
In the following,  $K$ , FATHER, TREE and BACK are defined as previously. We also have two new variables FORWARD and CROSS (their meanings are obvious) and

$$L(x) = \begin{cases} 1 & \text{if } x \text{ is completely scanned} \\ 0 & \text{otherwise.} \end{cases}$$

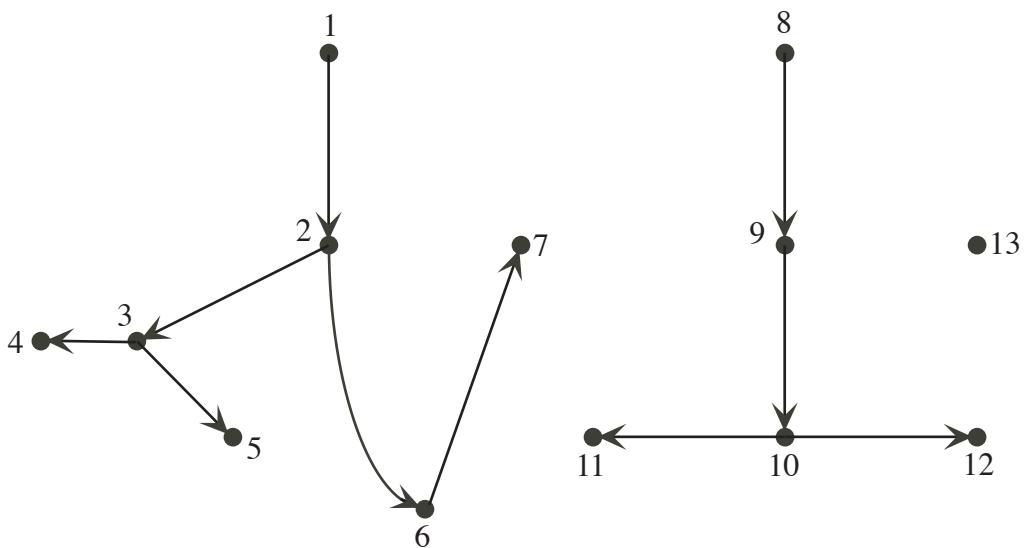
### Depth-First Search for Digraphs:

1. Set TREE  $\leftarrow \emptyset$ , FORWARD  $\leftarrow \emptyset$ , BACK  $\leftarrow \emptyset$ , CROSS  $\leftarrow \emptyset$  and  $i \leftarrow 1$ . For every vertex  $x$  in  $G$ , set FATHER( $x$ )  $\leftarrow 0$ ,  $K(x) \leftarrow 0$  and  $L(x) \leftarrow 0$ .
2. Choose a vertex  $r$  such that  $K(r) = 0$  and set  $\text{DFN}(r) \leftarrow i$ ,  $K(r) \leftarrow 1$  and  $u \leftarrow r$ .
3. If every arc coming out of  $u$  has already been examined, then set  $L(u) \leftarrow 1$  and go to step #5. Otherwise, choose an unexamined arc  $e = (u, v)$ .
4. Label the arc  $e$  examined.
  - 4.1 If  $K(v) = 0$ , then set  $i \leftarrow i + 1$ ,  $\text{DFN}(v) \leftarrow i$ , TREE  $\leftarrow \text{TREE} \cup \{e\}$ ,  $K(v) \leftarrow 1$ , FATHER( $v$ )  $\leftarrow u$  and  $u \leftarrow v$ . Go to step #3.
  - 4.2 If  $K(v) = 1$  and  $\text{DFN}(v) > \text{DFN}(u)$ , then set FORWARD  $\leftarrow \text{FORWARD} \cup \{e\}$  and go to step #3.
  - 4.3 If  $K(v) = 1$  and  $\text{DFN}(v) < \text{DFN}(u)$  and  $L(v) = 0$ , then set BACK  $\leftarrow \text{BACK} \cup \{e\}$  and go to step #3.
  - 4.4 If  $K(v) = 1$  and  $\text{DFN}(v) < \text{DFN}(u)$  and  $L(v) = 1$ , then set CROSS  $\leftarrow \text{CROSS} \cup \{e\}$  and go to step #3.
5. If FATHER( $u$ )  $\neq 0$ , then set  $u \leftarrow \text{FATHER}(u)$  and to go step #3.
6. If there is a vertex  $r$  such that  $K(r) = 0$ , then set  $i \leftarrow i + 1$  and go to step #2.
7. Stop.

**Example.** DFS in the following digraph starts from a root in the upper left corner and proceeds like this (back edges are marked with one line, cross edges are marked with two lines and forward edges are marked with three lines):



The corresponding DFS forest is



**Theorem 5.1.** *If a depth-first search in a quasi-strongly connected digraph starts from one of its roots, then the DFS forest is a directed tree. In particular, the DFS forest of a strongly connected digraph is a directed tree no matter where the search starts from.*

*Proof.* Let us prove by contradiction and consider the counter hypothesis: The DFS forest  $T$  resulted from a DFS in a quasi-strongly connected digraph  $G$  that began from root  $r$  is not a directed tree.

Since  $T$  is a directed forest, the component  $T_1$  of  $T$  which has the root  $r$  does not contain some vertex  $v$  of  $G$ . On the other hand, there is a directed path from  $r$  to  $v$ . We choose the last vertex  $u$  on this path which is in  $T_1$  and the arc  $e = (u, v)$ . Since the vertex  $w$  is not in  $T_1$ , the edge  $e$  is not a tree edge, a back edge nor a forward edge. Then, it must be a cross edge. Because the search began at  $r$ , the vertex  $w$  has to be in  $T_1$  ( $\checkmark$ ).

Strongly connected digraphs are also quasi-strongly connected and any vertex can be chosen as a root.  $\square$

*Breadth-first search*, BFS, is related to DFS. Let us consider a connected graph  $G$ .

#### Breadth-First Search for Graphs:

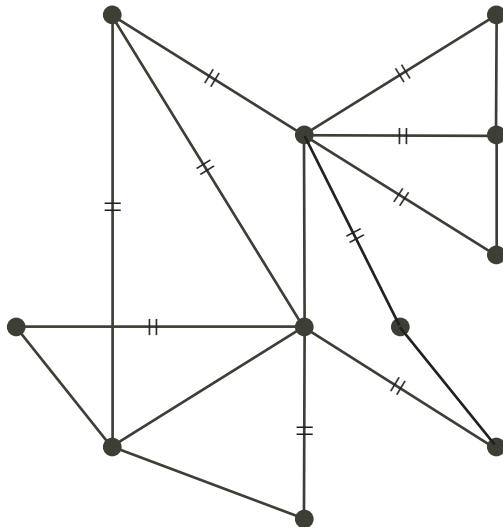
1. In the beginning, no vertex is labeled. Set  $i \leftarrow 0$ .
2. Choose a (unlabeled) starting vertex  $r$  (root) and label it with  $i$ .
3. Search the set  $J$  of vertices that are not labeled and are adjacent to some vertex labeled with  $i$ .
4. If  $J \neq \emptyset$ , then set  $i \leftarrow i + 1$ . Label the vertices in  $J$  with  $i$  and go to step #3.
5. (Only for disconnected graphs so we can jump from one component to another.) If a vertex is unlabeled, then set  $i \leftarrow 0$  and go to step #2.
6. Stop.

BFS also produces a spanning tree, called the *BFS tree*, when we take the edges

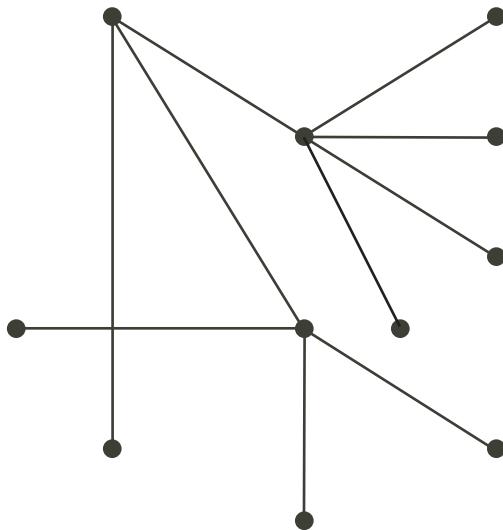
(vertex labeled with  $i$ , unlabeled vertex)

while forming  $J$ . One such *tree edge* exists for each vertex in  $J$ . We obtain the *directed BFS tree* by orienting the edges away from the labeled vertex to the unlabeled vertex. BFS as presented above does not however orient every edge in the graph. Obviously, the label of a vertex is the length of the shortest path from the root to it, in other words, the *distance* from the root.

**Example.** *BFS in the graph we had in the previous example starts at a root in the upper left corner and proceeds as follows. (Tree edges are marked with two cross lines.)*



The corresponding BFS tree is



We obtain the directed BFS tree by orienting the branches away from the root.

BFS in a digraph  $G$  is very similar to what we just did.

#### **Breadth-First Search for Digraphs:**

1. In the beginning, no vertex is labeled. Set  $i \leftarrow 0$ .
2. Choose an unlabeled starting vertex  $r$  (root) and label it with  $i$ .
3. Search the set  $J$  of terminal vertices of arcs whose initial vertices have been labeled with  $i$  but whose terminal vertices have not been labeled.
4. If  $J \neq \emptyset$ , then set  $i \leftarrow i + 1$ . Label the vertices in  $J$  with  $i$  and go to step #3.
5. If not all vertices have been labeled, then set  $i \leftarrow 0$  and go to step #2.
6. Stop.

BFS in a digraph produces a BFS forest (directed forest) when we take the examined arcs

(vertices labeled with  $i$ , unlabeled vertices)

while forming  $J$ . One such *tree edge* exists for each vertex in  $J$ .

**Remark.** In addition, BFS can be modified to sort the arcs like DFS.

## 5.4 The Lightest Path: Dijkstra's Algorithm

**Problem.** The edges of a (di)graph are given non-negative weights. The weight of a path is the sum of the weights of the path traversed. We are to find the lightest (directed) path in the (di)graph from vertex  $u$  to vertex  $v$  ( $\neq u$ ) if the path exists (sometimes also called the shortest path). We should state if such path does not exist.

Obviously, we can assume that we do not have any loops or parallel edges. Otherwise, we simply remove the loops and choose the edge with the lowest weight out of the parallel edges. From now on, we only consider directed graphs. Undirected graphs can be treated in the same way by replacing an edge with two arcs in opposite directions with the same weight.

We denote  $\alpha(r, s)$  as the weight of the arc  $(r, s)$ . *Dijkstra's Algorithm* marks the vertices as permanent or temporary vertices. The label of a vertex  $r$  is denoted  $\beta(r)$  and we define

$$\gamma(r) = \begin{cases} 1 & \text{if the label is permanent} \\ 0 & \text{if the label is temporary.} \end{cases}$$

A permanent label  $\beta(r)$  expresses the weight of the lightest directed  $u-r$  path. A temporary label  $\beta(r)$  gives an upper limit to this weight (can be  $\infty$ ). Furthermore, we denote:

$$\pi(r) = \begin{cases} \text{the predecessor of vertex } r \text{ on the lightest directed } u-r \text{ path if such a path exists} \\ 0 \text{ otherwise,} \end{cases}$$

so we can construct the directed path with the lowest weight.

### Dijkstra's Algorithm:

1. Set  $\beta(u) \leftarrow 0$  and  $\gamma(u) \leftarrow 1$ . For all other vertices  $r$ , set  $\beta(r) \leftarrow \infty$  and  $\gamma(r) \leftarrow 0$ . For all vertices  $r$ , we set  $\pi(r) \leftarrow 0$ . Furthermore, set  $w \leftarrow u$ .
2. For every arc  $(w, r)$ , where  $\gamma(r) = 0$  and  $\beta(r) > \beta(w) + \alpha(w, r)$ , set

$$\beta(r) \leftarrow \beta(w) + \alpha(w, r) \quad \text{and} \quad \pi(r) \leftarrow w.$$

3. Find a vertex  $r^*$  for which  $\gamma(r^*) = 0$ ,  $\beta(r^*) < \infty$  and

$$\beta(r^*) = \min_{\gamma(r)=0} \{\beta(r)\}.$$

Set

$$\gamma(r^*) \leftarrow 1 \quad \text{and} \quad w \leftarrow r^*.$$

If there is no such vertex  $r^*$ , a directed  $u-v$  path does not exist and we stop.

4. If  $w \neq v$ , then go to step #2.
5. Stop.

We see that the algorithm is correct as follows. We denote (for every step):

$$\begin{aligned} V_1 &= \{\text{permanently labeled vertices}\} \\ V_2 &= \{\text{temporarily labeled vertices}\}. \end{aligned}$$

( $\langle V_1, V_2 \rangle$  is a cut with the completely scanned vertices on one side and other vertices on the other side.)

**Statement.** *The label  $\beta(r)$  of the vertex  $r$  in  $V_1$  is the weight of the lightest directed  $u-r$  path and  $\pi(r)$  is the predecessor of  $r$  on such a path.*

*Proof.* After step #2, the temporary label of  $r$  is always the weight of a directed  $u-r$  path with the lowest weight whose vertices are in  $V_1$  except for  $r$  ( $= \infty$  if there is no such path), and  $\pi(r)$  is a predecessor of  $r$  on this path (or  $= 0$ ). This is because (two cases):

- Before step #2,  $\beta(r) = \infty$ . The only "new" vertex in  $V_1$  is now  $w$  so every possible directed  $u-r$  path has to visit  $w$ . If there is no such path, then the case is obvious ( $\beta(r)$  stays at  $\infty$  and  $\pi(r)$  stays at zero). Let us assume that we have the (lightest) directed  $u-r$  path that contains only vertices of  $V_1$  and  $r$  as well. In particular,  $w$  is included. The subpath from  $u$  to  $w$  has of course the lowest weight. We consider the vertex  $s$  ( $\in V_1$ ) which is the predecessor of  $r$  on the directed  $u-r$  path. If  $s = w$ , then the case is clear. If  $s \neq w$ , then  $s$  has been a  $w$  before, in which case  $\beta(r)$  can not be  $= \infty$  (step #2) (✓).
- Before step #2,  $\beta(r) < \infty$ . Then,  $\beta(r)$  is the weight of the lightest directed  $u-r$  path whose vertices are in  $V_1 - \{w\}$  except for  $r$ . The only "new" vertex in  $V_1$  is  $w$  so every possible lighter directed  $u-r$  path has to visit  $w$ . If there is no such path, then the case is obvious ( $\beta(r)$  and  $\pi(r)$  remain unchanged). Let us assume that we have a (lighter) directed  $u-r$  path that contains only vertices of  $V_1$  and  $r$  as well. In particular,  $w$  is included. The subpath from  $u$  to  $w$  has of course the lowest weight. We consider the vertex  $s$  ( $\in V_1$ ) which is the predecessor of  $r$  on the directed  $u-r$  path. If  $s = w$ , then the case is clear. If  $s \neq w$ , then  $s$  is in  $V_1 - \{w\}$ . Since  $s$  has been a  $w$  before, there is a lightest directed  $u-s$  path that does not contain  $w$  (otherwise, we should have chosen  $r^*$  in step #3 to be some predecessor of  $s$  on the directed  $u-w-s$  path). Then, we get a directed  $u-r$  path with a lower weight that contains  $r$  and only vertices in  $V_1 - \{w\}$  (✓).

The permanent label is the weight we seek because of the minimization in step #3 and  $\pi(r)$  gives a predecessor of  $r$  as we claimed.  $\square$

At the end of the algorithm, vertex  $v$  gets a permanent label or the process stops at step #3 (which means a directed  $u-v$  path does not exist). The directed path with the lowest weight can be obtained by starting from the vertex  $v$  and finding the predecessors by using the label  $\pi$ .

If we replace step #4 by

- 4'. Go to step #2.

and continue the process until it stops at step #3, we get

$$\beta(w) = \begin{cases} 0 & \text{if } w = u \\ \text{the weight of the lightest directed } u-w \text{ path if there is one} \\ \infty & \text{otherwise} \end{cases}$$

and

$$\pi(w) = \begin{cases} \text{the predecessor of } w \text{ on the lightest directed } u-w \text{ path if there is one and } w \neq u \\ 0 \text{ otherwise.} \end{cases}$$

**Remark.** Dijkstra's algorithm may fail if there are negative weights. These cases are investigated in the next section.

## 5.5 The Lightest Path: Floyd's Algorithm

**Problem.** We are to find the lightest path from vertex  $u$  to vertex  $v$  ( $\neq u$ ) in a digraph or to show that there is no such path when the arcs of the digraph have been assigned arbitrary weights. Note that the weight of a directed path is the sum of the weights of the arcs traversed.

Obviously, we can assume there are no loops or parallel arcs. Otherwise, we simply remove the loops and choose the arc with the lowest weight out of the parallel arcs. Floyd's Algorithm only works for digraphs. We write the weight of  $(x, y)$  as  $\alpha(x, y)$  and construct the *weight matrix*  $\mathbf{W} = (w_{ij})$  where

$$w_{ij} = \begin{cases} \alpha(v_i, v_j) \text{ if there is an arc } (v_i, v_j) \\ \infty \text{ otherwise.} \end{cases}$$

(Once again,  $V = \{v_1, \dots, v_n\}$  is the vertex set of the digraph.) Floyd's Algorithm is similar to Warshall's Algorithm. It only works if the digraph has no negative cycles, i.e. no directed circuit in the digraph has a negative weight. In this case, the lightest directed path is the lightest directed walk.

Floyd's Algorithm constructs a sequence of matrices  $\mathbf{W}_0, \mathbf{W}_1, \dots, \mathbf{W}_n$  where  $\mathbf{W}_0 = \mathbf{W}$  and

$$\begin{aligned} (\mathbf{W}_k)_{ij} &= \text{weight of the lightest directed } v_i-v_j \text{ path,} \\ &\quad \text{where there are only vertices } v_1, \dots, v_k \text{ on the path besides } v_i \text{ and } v_j \\ &\quad (= \infty \text{ if there is no such path).} \end{aligned}$$

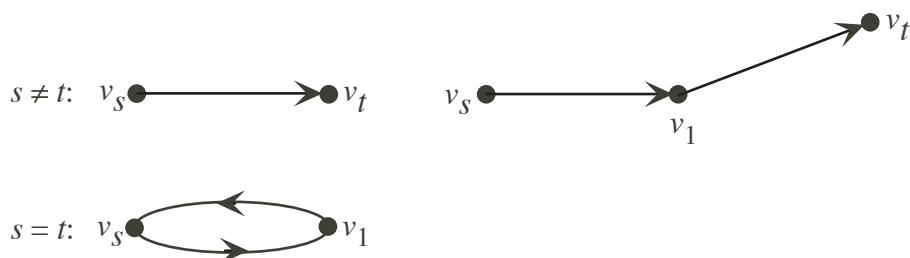
**Statement.** When  $\mathbf{W}_k$  is computed from  $\mathbf{W}_{k-1}$  by the formula

$$(\mathbf{W}_k)_{st} = \min\{(\mathbf{W}_{k-1})_{st}, (\mathbf{W}_{k-1})_{sk} + (\mathbf{W}_{k-1})_{kt}\},$$

then we get the previously mentioned sequence of weight matrices. If the digraph has negative cycles, then the sequence is correct up to the point when one of the diagonal elements turns negative for the first time.

*Proof.* We use induction on  $k$ .

Induction Basis:  $k = 1$ . Since the digraph is loopless, the diagonal elements of  $\mathbf{W}_0$  can only be  $\infty$  and the lightest directed path (if there is one) is one of the following, and the statement is obvious:



Induction Hypothesis: The statement is true for  $k < \ell$ . ( $\ell \geq 2$ )

Induction Statement: The statement is true for  $k = \ell$ .

Induction Statement Proof: The diagonal elements of  $\mathbf{W}_{\ell-1}$  have to be nonnegative ( $\infty$  is permitted) for us to get this  $k$ . Let us consider the case where  $s \neq t$ . (The case  $s = t$  is analogous.) We have five cases:

- Vertex  $v_\ell$  is on the lightest directed path but it is not  $v_s$  or  $v_t$ , i.e.  $\ell \neq s, t$ . Let us consider the directed subpath from  $v_s$  to  $v_\ell$  whose vertices other than  $v_s$  and  $v_\ell$  are in  $\{v_1, \dots, v_{\ell-1}\}$ . Suppose the lightest directed  $v_s-v_\ell$  path of this kind has common vertices with the directed subpath from  $v_\ell$  to  $v_t$  other than  $v_\ell$  itself, e.g.  $v_p$ . The directed  $v_s-v_p-v_\ell-v_t$  walk we get would be lighter than the original directed  $v_s-v_t$  path. By removing cycles, we would get a directed  $v_s-v_t$  path that would be lighter and would only contain  $v_s$  as well as  $v_t$  and the vertices  $v_1, \dots, v_\ell$  ( $\checkmark$ ). (We have to remember that weights of cycles are not negative!) Therefore, the directed subpath from  $v_s$  to  $v_\ell$  is the lightest directed  $v_s-v_\ell$  path which contains the vertices  $v_1, \dots, v_{\ell-1}$  as well as  $v_s$  and  $v_\ell$ . Similarly, the directed subpath from  $v_\ell$  to  $v_t$  is the lightest directed  $v_\ell-v_t$  path which contains the vertices  $v_1, \dots, v_{\ell-1}$  as well as  $v_t$  and  $v_\ell$ . Now, we use the Induction Hypothesis:

$$(\mathbf{W}_\ell)_{st} < (\mathbf{W}_{\ell-1})_{st}$$

(check the special case  $(\mathbf{W}_{\ell-1})_{st} = \infty$ ) and

$$(\mathbf{W}_\ell)_{st} = (\mathbf{W}_{\ell-1})_{s\ell} + (\mathbf{W}_{\ell-1})_{\ell t}.$$

- The directed  $v_s-v_t$  path with the lowest weight exists and  $v_\ell = v_s$ . By the Induction Hypothesis,  $(\mathbf{W}_\ell)_{st} = (\mathbf{W}_{\ell-1})_{st}$  and

$$(\mathbf{W}_{\ell-1})_{s\ell} + (\mathbf{W}_{\ell-1})_{\ell t} = (\mathbf{W}_{\ell-1})_{\ell\ell} + (\mathbf{W}_{\ell-1})_{\ell t} \geq (\mathbf{W}_{\ell-1})_{\ell t} = (\mathbf{W}_{\ell-1})_{st},$$

since  $(\mathbf{W}_{\ell-1})_{\ell\ell} \geq 0$  (possibly  $= \infty$ ).

- The directed  $v_s-v_t$  path exists and  $v_\ell = v_t$ . By the Induction Hypothesis,  $(\mathbf{W}_\ell)_{st} = (\mathbf{W}_{\ell-1})_{st}$  and

$$(\mathbf{W}_{\ell-1})_{s\ell} + (\mathbf{W}_{\ell-1})_{\ell t} = (\mathbf{W}_{\ell-1})_{s\ell} + (\mathbf{W}_{\ell-1})_{\ell\ell} \geq (\mathbf{W}_{\ell-1})_{s\ell} = (\mathbf{W}_{\ell-1})_{st},$$

since  $(\mathbf{W}_{\ell-1})_{\ell\ell} \geq 0$  (possibly  $= \infty$ ).

- The lightest directed  $v_s-v_t$  path exists but  $v_\ell$  is not on the path. Now, we construct the lightest directed  $v_s-v_\ell$  path and the lightest  $v_\ell-v_t$  path which, in addition to the end vertices, contain only vertices  $v_1, \dots, v_{\ell-1}$ , if it is possible. By combining these two paths, we get a directed  $v_s-v_t$  walk. By removing possible cycles from this walk, we get an as light or even lighter  $v_s-v_t$  path, which only contains vertices  $v_1, \dots, v_\ell$  as well as  $v_s$  and  $v_t$ . (We have to remember that weights of cycles are not negative!) Therefore, this is a case where

$$(\mathbf{W}_{\ell-1})_{s\ell} + (\mathbf{W}_{\ell-1})_{\ell t} \geq (\mathbf{W}_{\ell-1})_{st}$$

and the equation in the statement gives the right result. If there is no directed  $v_s-v_\ell$  path or  $v_\ell-v_t$  path, then it is obvious.

- The lightest directed  $v_s-v_t$  path does not exist. Then,  $(\mathbf{W}_\ell)_{st} = \infty$  and  $(\mathbf{W}_{\ell-1})_{st} = \infty$ . On the other hand, at least one of the elements  $(\mathbf{W}_{\ell-1})_{s\ell}$  or  $(\mathbf{W}_{\ell-1})_{\ell t}$  is  $= \infty$  because otherwise we would get a directed  $v_s-v_t$  path by combining the  $v_s-v_\ell$  path with the  $v_\ell-v_t$  path as well as removing all possible cycles ( $\checkmark$ ).  $\square$

Floyd's Algorithm also constructs another sequence of matrices  $\mathbf{Z}_0, \dots, \mathbf{Z}_n$  in which we store the lightest directed paths in the following form

$$(\mathbf{Z}_k)_{ij} = \begin{cases} \ell \text{ where } v_\ell \text{ is the vertex following } v_i \text{ on the lightest directed} \\ \quad v_i-v_j \text{ path containing only vertices } v_i \text{ and } v_j \text{ as well as } v_1, \dots, v_k \\ \quad (\text{if such a path exists}) \\ 0 \text{ otherwise.} \end{cases}$$

Obviously,

$$(\mathbf{Z}_0)_{ij} = \begin{cases} j \text{ if } (\mathbf{W})_{ij} \neq \infty \\ 0 \text{ otherwise.} \end{cases}$$

The matrix  $\mathbf{Z}_k$  ( $k \geq 1$ ) of the sequence can be obtained from the matrix  $\mathbf{Z}_{k-1}$  by

$$(\mathbf{Z}_k)_{ij} = \begin{cases} (\mathbf{Z}_{k-1})_{ik} \text{ if } (\mathbf{W}_{k-1})_{ik} + (\mathbf{W}_{k-1})_{kj} < (\mathbf{W}_{k-1})_{ij} \\ (\mathbf{Z}_{k-1})_{ij} \text{ otherwise,} \end{cases}$$

so the sequence can be constructed with the sequence  $\mathbf{W}_0, \mathbf{W}_1, \dots, \mathbf{W}_n$  at the same time. Finally, Floyd's Algorithm is presented in the following pseudocode. We have added a part to test if there are negative elements on the diagonal and the construction of the  $\mathbf{Z}_0, \dots, \mathbf{Z}_n$  sequence of matrices.

```

procedure Floyd
begin
     $\mathbf{W} := \mathbf{W}_0$ 
     $k := 0$ 
    for  $i := 1$  to  $n$  do
        for  $j := 1$  to  $n$  do
            if  $(\mathbf{W})_{ij} = \infty$  then
                 $(\mathbf{Z})_{ij} := 0$ 
            else
                 $(\mathbf{Z})_{ij} := j$ 
            fi
        od
    od
    while  $k < n$  and Test( $\mathbf{W}$ ) do
        Iteration( $\mathbf{W}, \mathbf{Z}, k$ )
    od
end

subprocedure Test( $\mathbf{W}$ )
begin
    for  $i := 1$  to  $n$  do
        if  $(\mathbf{W})_{ii} < 0$  then
            return FALSE
        fi
    od
    return TRUE
end
```

```

subprocedure Iteration( $\mathbf{W}, \mathbf{Z}, k$ )
begin
     $k := k + 1$ 
    for  $i := 1$  to  $n$  do
        for  $j := 1$  to  $n$  do
            if  $(\mathbf{W})_{ik} + (\mathbf{W})_{kj} < (\mathbf{W})_{ij}$  then
                 $(\mathbf{W})_{ij} := (\mathbf{W})_{ik} + (\mathbf{W})_{kj}$ 
                 $(\mathbf{Z})_{ij} := (\mathbf{Z})_{ik}$ 
            fi
        od
    od
end

```

## 5.6 The Lightest Spanning Tree: Kruskal's and Prim's Algorithms

**Problem.** We have to find the spanning tree with the lowest weight of a connected graph if the edges of the graph have been weighted arbitrarily and the weight of a tree is the sum of all the weights of the branches.

Obviously, we can assume that the graph  $G = (V, E)$  is nontrivial and simple. Otherwise, we simply remove the loops and choose the edge with the lowest weight out of the parallel edges. We denote the weight of the edge  $e$  as  $\alpha(e)$  and the weight of the spanning tree  $T$  as  $\gamma(T)$ . As usual, we write the number of vertices as  $n$ , number of edges as  $m$ ,  $V = \{v_1, \dots, v_n\}$  and  $E = \{e_1, \dots, e_m\}$ .

The *distance* between two spanning trees  $T_1$  and  $T_2$  of  $G$  is

$$n - 1 - \#(T_1 \cap T_2) =_{\text{def.}} d(T_1, T_2),$$

where  $\#(T_1 \cap T_2)$  is the number of edges in the intersection of  $T_1$  and  $T_2$ . Obviously,  $d(T_1, T_2) = 0$  if and only if  $T_1 = T_2$ . If  $d(T_1, T_2) = 1$ , then  $T_1$  and  $T_2$  are *neighboring trees*.

The spanning tree  $T$  of  $G$  is *cut minimal* if the weights of the edges of the fundamental cut set determined by the branch  $b$  are  $\geq \alpha(b)$  for every branch  $b$ . Similarly, the spanning tree  $T$  is *circuit minimal* if the edges of the fundamental circuits are  $\leq \alpha(c)$  for every link  $c$  in the cospanning tree  $T^*$ . The spanning tree  $T$  is *locally minimal* if  $\gamma(T) \leq \gamma(T')$  for every neighboring tree  $T'$  of  $T$ .

**Lemma.** The following three conditions are equivalent for the spanning tree  $T$ :

- (i)  $T$  is cut minimal.
- (ii)  $T$  is circuit minimal.
- (iii)  $T$  is locally minimal.

*Proof.* (i) $\Rightarrow$ (ii): Let us assume  $T$  is cut minimal and let us consider a fundamental circuit  $C$  of  $G$  corresponding to the link  $c$  of the cospanning tree  $T^*$ . Other than  $c$ , the branches in  $C$  are branches of  $T$ . Every such branch  $b$  defines a fundamental cut set of  $T$ , which also contains  $c$  (Theorem 2.7). Hence  $\alpha(b) \leq \alpha(c)$ .

(ii) $\Rightarrow$ (iii): Let us assume that  $T$  is circuit minimal and let us consider a neighboring tree  $T'$  of  $T$ .  $T'$  has (exactly one) branch  $e'$  which is not in  $T$ , i.e.  $e'$  is a link of  $T^*$ . We examine the

fundamental circuit  $C$  defined by  $e'$ . Not all edges of  $C$  are in  $T'$ . We choose an edge  $e$  in  $C$  that is not in  $T'$ . Then,  $e$  is a branch of  $T$  (actually the only branch of  $T$  that is not in  $T'$ ). Now, we remove  $e$  out of  $T$  and add  $e'$  to  $T$ . The result has to be  $T'$ . Because of circuit minimality,  $\alpha(e') \geq \alpha(e)$ , i.e.  $\gamma(T') \geq \gamma(T)$ .

(iii) $\Rightarrow$ (i): We consider the locally minimal spanning tree  $T$ . We take an arbitrary branch  $b$  from  $T$  corresponding to a fundamental cut set  $I$  and an arbitrary link  $c \neq b$  in  $I$ . Then,  $b$  belongs to the fundamental circuit of  $T$  defined by  $c$  (Theorem 2.8). By removing the branch  $b$  from  $T$  and adding the edge  $c$  to  $T$ , we get the neighboring tree  $T'$  of  $T$ . Because of local minimality,  $\gamma(T) \leq \gamma(T')$ , i.e.  $\alpha(c) \geq \alpha(b)$ .  $\square$

The spanning tree  $T$  is *minimal* if it has the lowest possible weight.

**Theorem 5.2.** *The following three conditions are equivalent for the spanning tree  $T$ :*

- (i)  $T$  is cut minimal.
- (ii)  $T$  is circuit minimal.
- (iii)  $T$  is minimal.

*Proof.* By the lemma above, (i) and (ii) are equivalent. A minimal spanning tree is obviously locally minimal. Thus, it suffices to prove that a cut minimal spanning tree is also minimal. We will prove by contradiction and consider the counter hypothesis: There is a cut minimal spanning tree  $T$  which is not minimal. Let us consider the minimal spanning tree  $T'$  and choose  $T$  and  $T'$  so that the distance  $d(T, T')$  is as small as possible. By the lemma,  $d(T, T') > 1$ .

$T$  has a branch  $e$  which is not in  $T'$ , i.e. it is a link of  $(T')^*$ . We label the fundamental cut set of  $T$  defined by  $e$  as  $I$  and the fundamental circuit of  $T'$  defined by  $e$  as  $C$ . In the intersection  $I \cap C$ , there are also other edges besides  $e$  (Theorem 2.6). We choose such an edge  $e'$ . Then,  $e'$  is a link of  $T^*$  and a branch of  $T'$ . Since  $T$  is cut minimal,  $\alpha(e') \geq \alpha(e)$ . Since  $T'$  is (circuit) minimal,  $\alpha(e') \leq \alpha(e)$ . Therefore,  $\alpha(e') = \alpha(e)$ . By removing  $e'$  from  $T'$  and adding  $e$  to  $T'$ , we get a minimal spanning tree  $T''$  which has the same weight as  $T'$ . However,  $d(T, T'') < d(T, T')$ .  $\checkmark$   $\square$

In *Kruskal's Algorithm*, the edges of the graph  $G$  (and their weights) are listed as  $e_1, \dots, e_m$ . The algorithm constructs a circuit minimal spanning tree by going through the list to take some edges to form the tree. This is especially effective if the edges are sorted in ascending order by weight.

In the *dual form of Kruskal's Algorithm*, we construct a cut minimal spanning tree by going through the list of edges to take some edges to form the cospanning tree. Once again, this is especially effective if the edges are sorted in descending order by weight.

In all, we get four different versions of Kruskal's Algorithm. (We have to remember that the subgraph induced by the edge set  $A$  is written as  $\langle A \rangle$ .)

### Kruskal's Algorithm No. 1

Here we assume that the edges are given in *ascending order* by weight.

1. Set  $k \leftarrow 1$  and  $A \leftarrow \emptyset$ .
2. If  $e_k$  does not form a circuit with the edges in  $A$ , then set  $A \leftarrow A \cup \{e_k\}$  as well as  $k \leftarrow k + 1$  and go to step #4.
3. If  $e_k$  forms a circuit with the edges in  $A$ , then set  $k \leftarrow k + 1$  and go to step #4.

4. If  $(V, A)$  is not a tree, then go to step #2. Otherwise stop and output the spanning tree  $T = \langle A \rangle$ .

Whenever we leave out an edge from  $A$  (step #3), its end vertices are already connected in  $A$ . Thus, the vertices of  $G$  are connected in  $T$  as they are in  $G$ . Since  $T$  is obviously circuitless (step #3), it is also a spanning tree of  $G$ . At each stage, the branches of the fundamental circuit defined by the link belonging to  $T^*$  (step #3) are predecessors of that link in the list. Hence,  $T$  is circuit minimal and thus minimal.

**Remark.** *In every step, the branches and links are permanent. We do not have to know the edges beforehand as long as we process them one by one in ascending order. The rank of the graph (number of branches in a spanning tree) is then required beforehand so we know when to stop.*

### Kruskal's Algorithm No. 2

Here we assume the edges are given in an *arbitrary order*.

1. Set  $k \leftarrow 1$  and  $A \leftarrow \emptyset$ .
2. If  $\langle A \cup \{e_k\} \rangle$  contains no circuits, then set  $A \leftarrow A \cup \{e_k\}$  as well as  $k \leftarrow k + 1$  and go to step #4.
3. If  $\langle A \cup \{e_k\} \rangle$  contains a circuit  $C$ , then choose the edge with the largest weight  $e$  in  $C$  (if there are more than one, take any), set  $A \leftarrow (A \cup \{e_k\}) - \{e\}$  as well as  $k \leftarrow k + 1$  and go to step #4.
4. If  $k \leq m$ , then go to step #2. Otherwise, stop and output the spanning tree  $T = \langle A \rangle$ .

Whenever we leave out an edge from  $A$  (step #3), its end vertices are already connected in  $A$ . Thus, the vertices of  $G$  are connected in  $T$  as they are in  $G$ . Since  $T$  is obviously circuitless (step #3), it is a spanning tree of  $G$ .

We see that  $T$  is circuit minimal (and minimal) by the following logic. During the whole process,  $\langle A \rangle$  is a forest by step #4. In addition, if  $u$  and  $w$  are connected in  $\langle A \rangle$  at some point, then they are also connected afterwards. The  $u-w$  path in  $\langle A \rangle$  is unique but it can change to another path later in step #3. Nevertheless, whenever this change occurs, the maximum value of the weights of the edges of the path can not increase anymore. Every link  $c$  of  $T^*$  has been removed from  $A$  in step #3. Then, the weight of  $c$  is at least as large as the weights of the other edges in  $C$ . After we have gone through step #3, the only connected end vertices of  $c$  in  $\langle A \rangle$  have to go through the remaining edges of  $C$ . The final connection between the end vertices of  $c$  in  $T$  goes through the edges of the fundamental circuit defined by  $c$ . Therefore, the weights of the edges of this fundamental circuit are  $\leq \alpha(c)$ .

**Remark.** *In each step, the links ( $e$  in step #3) are permanent and the branches are not. We do not have to know the edges beforehand as long as we process them one by one. However, we need to know the nullity of the graph (number of links in a cospanning tree) so that we know when to stop. The algorithm can also be used to update a minimal spanning tree if we add edges to the graph or decrease their weight.*

### Kruskal's Algorithm No. 3

Here we assume the edges are given in *descending order* by weight.

1. Set  $A \leftarrow E$  and  $k \leftarrow 1$ .
2. If  $(V, A - \{e_k\})$  is connected, then set  $A \leftarrow A - \{e_k\}$  as well as  $k \leftarrow k + 1$  and go to step #4.
3. If  $(V, A - \{e_k\})$  is disconnected, then set  $k \leftarrow k + 1$  and go to step #4.
4. If  $(V, A)$  is not a tree, then we go to step #2. Otherwise we stop and output the spanning tree  $T = (V, A)$ .

$T$  is obviously connected because  $(V, A)$  is connected everytime we go to step #4. On the other hand,  $T$  is circuitless because if the circuit  $C$  is in  $T$  and the edge  $c$  is in the circuit, then  $c$  is removed from  $A$  in step #2 when  $e_k = c$  ( $\checkmark$ ). Thus,  $T$  is a spanning tree of  $G$ . In each step, the links of the fundamental cut set defined by the branch belonging to  $T$  (step #3) are predecessors of that branch in the list. Hence,  $T$  is cut minimal and it is thus minimal.

**Remark.** *In each step, the branches and links are permanent. We have to know the edges beforehand. On the other hand, we do not have to know their weights as long as we get them one by one in descending order.*

### Kruskal's Algorithm No. 4

Here we assume the edges are given in an *arbitrary order*.

1. Set  $A \leftarrow E$  and  $k \leftarrow 1$ .
2. If  $(V, A - \{e_k\})$  is connected, then set  $A \leftarrow A - \{e_k\}$  as well as  $k \leftarrow k + 1$  and go to step #4.
3. If  $(V, A - \{e_k\})$  is disconnected, then it has two components. The corresponding vertex sets form a cut  $\langle V_1, V_2 \rangle$ . We interpret it as an edge set and choose the edge  $e$  with the lowest weight in  $\langle V_1, V_2 \rangle$  (if there are more than one, take any). Set  $A \leftarrow (A - \{e_k\}) \cup \{e\}$  as well as  $k \leftarrow k + 1$  and go to step #4.
4. If  $k \leq m$ , then go to step #2. Otherwise stop and output the spanning tree  $T = (V, A)$ .

$T$  is obviously connected because  $(V, A)$  is connected everytime we go to step #4. (Take note that the connectivity is preserved everytime we go through step #3.) On the other hand,  $T$  is circuitless. If a circuit  $C$  of  $G$  ends up in  $T$  and  $c$  is the edge of  $C$ , which is first in the list, then  $c$  must be removed from  $A$  in step #2 when  $e_k = c$ . (Note that the edge removed from the circuit first can not be removed in step #3.) If  $c$  comes back later (in step #3), then it forms a cut set of  $(V, A)$  by itself in which case some other edge of  $C$  has been removed. By continuing this process, we see that all the edges of  $C$  can not be in  $A$  in the end ( $\checkmark$ ). Therefore,  $T$  is a spanning tree of  $G$ .

In addition,  $T$  is cut minimal and minimal because every branch  $b$  of  $T$  comes in in step #3. The links of the fundamental cut set defined by  $b$  are either edges of the cut  $\langle V_1, V_2 \rangle$  which is examined at that point or they are links of the cuts we examined later in step #3. Whenever an edge of this kind gets removed later in step #3, it is always compensated by edges that are heavier in weight than  $b$ . Those heavier edges are in the cut  $\langle V_1, V_2 \rangle$  which is examined at that time. Therefore, the weights of the fundamental cut set defined by  $b$  are  $\geq \alpha(b)$ .

**Remark.** In each step, the branches ( $e$  in step #3) are permanent and the links are not. We have to know the edges beforehand. We do not have to know the weights beforehand as long as we process them one by one. This algorithm can also be used for updating a minimal spanning tree if we remove edges from a graph or if we increase the weights of edges.

### Prim's Algorithm

In *Prim's Algorithm* (also known as *Jarnik's Algorithm*), we use the all-vertex incidence matrix of  $G$ . If we label the set of edges incident on vertex  $v$  as  $\Omega(v)$ , then we can get a list  $\Omega(v_1), \dots, \Omega(v_n)$ , i.e. the cuts defined by the vertices (interpreted as edge sets). In addition, we assign weights to the vertices.

The algorithm works in the same way as Dijkstra's Algorithm by constructing the spanning tree branch by branch. The variables are  $A$  (set of branches of the spanning tree we have at the time),  $B$  (set of vertices of the spanning tree we have at the time) and  $I$  (the cut interpreted as an edge set from which we choose the next branch).

#### Prim's Algorithm (First Version):

1. Choose a starting vertex  $r$  and set  $A \leftarrow \emptyset$ ,  $B \leftarrow \{r\}$  as well as  $I \leftarrow \Omega(r)$ .
2. Choose the lightest edge  $e$  from  $I$  (if there are more than one, choose any). Take the end vertex  $v$  of  $e$  that is not in  $B$ . Set  $A \leftarrow A \cup \{e\}$ ,  $B \leftarrow B \cup \{v\}$  as well as  $I \leftarrow I \oplus \Omega(v)$  and go to step #3. (Remember that  $\oplus$  denotes the symmetric difference operation between two sets, see page 12.)
3. If  $B \neq V$ , then go to step #2. Otherwise, stop and output the spanning tree  $T = (B, A) = \langle A \rangle$ .

Since the edge  $e$  was chosen from a cut,  $T$  is circuitless. On the other hand, because there is a path from  $r$  to every other vertex,  $T$  has every vertex of  $G$  and it is connected.  $T$  is thus a spanning tree. It is also minimal because

**Statement.** During the whole process,  $(B, A)$  is a subtree of some minimal spanning tree of  $G$ .

*Proof.* We use induction on  $\ell$ , the number of vertices in  $B$ .

Induction Basis:  $\ell = 1$ . The case is obvious because  $(B, A)$  is trivial.

Induction Hypothesis: The statement is true for  $\ell = k - 1$ . ( $k \geq 2$ )

Induction Statement: The statement is true for  $\ell = k$ .

Induction Statement Proof: In step #2, we can write  $A = A' \cup \{e\}$ , where  $e \in I'$  and  $B = B' \cup \{v\}$ .  $(B', A')$  is a subtree of some minimal spanning tree  $T_{\min}$  from the induction hypothesis. If  $e$  belongs to  $T_{\min}$ , then the case is clear. Otherwise, there is a fundamental circuit  $C$  in  $T_{\min} + e$  and there is another edge  $e'$  of  $I'$  in  $C$  (Theorem 2.6). Then,  $\alpha(e') \geq \alpha(e)$  and  $(T_{\min} + e) - e'$  is also a minimal spanning tree and  $(B, A)$  is its subtree (because  $T_{\min}$  is circuit minimal and  $\alpha(e') \leq \alpha(e)$ ).  $\square$

Often, we use one or two additional labels for the vertices to make Prim's Algorithm easier. In the next version of the algorithm, we will use two labels  $\pi(v)$  and  $\beta(v)$ , which are used to perform step #2 more effectively. The values of  $\pi$  are weights (up to  $\infty$ ) and the values of  $\beta$  are edges (or = 0). Otherwise, the algorithms works in the same way as before.

**Prim's Algorithm (Second Version):**

1. Choose a starting vertex  $r$  and set  $\pi(r) \leftarrow 0$ . For every other vertex  $v$ , set  $\pi(v) \leftarrow \infty$ . For every vertex  $v$ , set  $\beta(v) \leftarrow 0$  as well as  $A \leftarrow \emptyset$  and  $B \leftarrow \emptyset$ .
2. Choose a vertex  $u \notin B$  for which

$$\pi(u) = \min_{v \notin B} \{\pi(v)\}.$$

Set  $B \leftarrow B \cup \{u\}$ . If  $\beta(u) \neq 0$ , then set  $A \leftarrow A \cup \{\beta(u)\}$ .

3. Go through all the edges  $e = (u, v)$  where  $v \notin B$ . If  $\alpha(e) < \pi(v)$ , then set  $\pi(v) \leftarrow \alpha(e)$  and  $\beta(v) \leftarrow e$ .
4. If  $B \neq V$ , then go to step #2. Otherwise, stop and output the spanning tree  $T = (B, A) = \langle A \rangle$ .

## 5.7 The Lightest Hamiltonian Circuit (Travelling Salesman's Problem): The Annealing Algorithm and the Karp–Held Heuristics

**Problem.** *If it is possible, we are to find the Hamiltonian circuit with the lowest weight. A Hamiltonian circuit visits all the vertices of a graph. As usual, the weights of the edges have been assigned and the weight of a (directed) circuit is the sum of the weights of the edges traversed.*

Obviously, we can assume that the graph is nontrivial, connected (otherwise it would not be possible to get a Hamiltonian circuit) and simple. If not, then we simply remove all the loops and choose the edge with the lowest weight out of the parallel edges. As usual, we denote  $n$  as the number of vertices,  $m$  as the number of edges,  $V = \{v_1, \dots, v_n\}$  and  $E = \{e_1, \dots, e_m\}$ . We label the weight of an edge  $e = (v_i, v_j)$  as  $\alpha(e) = \alpha(v_i, v_j)$  and the weight of a Hamiltonian circuit  $H$  as  $\gamma(H)$ . We agree that the "first" vertex of a Hamiltonian circuit is  $v_1$ .

The same problem exists for directed graphs in which case we are looking for the directed Hamiltonian circuit with the lowest weight (known as the *Unsymmetric Travelling Salesman's Problem*).

The Travelling Salesman's Problem (TSP)<sup>1</sup> is an  $\mathcal{NP}$ -complete problem, read e.g. MEHLHORN for more information. Actually, even deciding the existence of a Hamiltonian circuit is an  $\mathcal{NP}$ -complete problem. Solving a small TSP takes a lot of time and larger problems take so much time that it is almost impossible to obtain accurate solutions. Therefore, many stochastic and approximation methods are used in practice. Then, we have to accept the possibility of inaccurate outcomes or even the lack of results.

---

<sup>1</sup>The name "Travelling Salesman's Problem" comes from an interpretation where the vertices of a graph are cities and the weights of the edges between the cities are travelling times. The salesman needs to visit every city in the shortest amount of time.

### The Annealing Algorithm

The *annealing algorithms* or *thermodynamic algorithms* have the following common features:

- (A) The system in question is always in some *state*  $s$ . The set of all states  $S$  is finite and known. In the TSP, a state is a Hamiltonian circuit.
- (B) Each state  $s$  has a *response*  $f(s)$ , which can be calculated in a timely fashion from the state. Our goal is to find a state whose response is near the minimum/maximum value. The response of a state of a TSP is the weight of a Hamiltonian circuit.
- (C) There is a procedure  $A_k$  which is used to move from state  $s$  to state  $A_k(s)$ .  $k$  is a parameter of the procedure which belongs to the set  $K$ .  $K$  can change during the procedure. The purpose is to move to certain states "near" the state  $s$  which are defined by the parameter  $k$ . By repeating the procedure with proper values of  $k$ , we should be able to move from any state to any other state. (In some cases, we can omit this last part.)
- (D) Every time we move from one state to another, we should be able to choose the parameter  $k$  quickly and randomly from  $K$ . In particular, the set  $K$  itself should be easily computable.
- (E) We should be able to quickly perform the procedure  $A_k$  given a value of  $k$ .
- (F) We should be able to find a starting state  $s_0$ . For the TSP, the starting state is a Hamiltonian circuit.

The algorithm is as follows:

#### **The Annealing Algorithm:**

1. Choose the starting state  $s_0$ , the *initial temperature*  $T_0$  and set  $s \leftarrow s_0$  as well as  $T \leftarrow T_0$ .
2. When we are in the state  $s$ , we randomly choose a parameter  $k \in K$  and compute  $s' = A_k(s)$ .
3. If  $f(s') \leq f(s)$ , then set  $s \leftarrow s'$  and go to step #5.
4. If  $f(s') > f(s)$ , then generate a random number  $r$  in the interval  $[0, 1]$ . If  $r \leq e^{\frac{f(s)-f(s')}{T}}$ , then set  $s \leftarrow s'$ . Thus, we accept a "worse" state with probability  $e^{\frac{f(s)-f(s')}{T}}$ . Note that the greater the temperature  $T$ , the greater the probability that we go "uphill".
5. If we have gone through a maximum total number of iterations, then we stop and output  $s$ . Otherwise, if we have gone through sufficiently many iterations of the procedure using temperature  $T$ , then we lower  $T$  by some rule and go to step #2.

**Remark.** The distribution of the probability  $p_{s'} = e^{\frac{f(s)-f(s')}{T}}$  used in step #4 is (apart from normalizing) a so-called maximum entropy distribution with the following condition on the expected value:

$$\sum_{\substack{s' = A_k(s) \\ k \in K \\ f(s') > f(s)}} p_{s'} f(s') = \mu$$

where  $\mu$  depends on  $T$  and  $s$ . The distribution is also called a Boltzman distribution and it is analogous to the distribution of the same name in Statistical Mechanics. Refer to courses in Physics and Information Theory for more information.

At first, we wait until the fluctuation in the states settles to a certain equilibrium (using the response  $f(s)$ ). After that, we lower the value of  $T$  a bit and wait again for the equilibrium. Then, we lower  $T$  again and so on. We continue this until the change in values of  $f(s)$  is sufficiently small or if we have ran out of time.

The operation  $A_k$  and the set  $K$  of the neighboring states depend on the problem. The state structure and the response function also depend on the problem. For the TSP, we still need to assign  $A_k$  and  $K$  for every situation. For this purpose, we take another parameter  $j$  and set  $j \leftarrow 2$ . In step #2, we update  $j$  in the following way:

$$j \leftarrow \begin{cases} j + 1 & \text{if } j < n \\ 2 & \text{otherwise.} \end{cases}$$

(Another way of choosing  $j$  in step #2 would be to choose it randomly out of  $\{2, \dots, n\}$ .) Furthermore, we choose

$$K = \{2, \dots, n\} - \{j\}.$$

$A_k$  is defined by the following operation (known as the *reversal*):

- If  $k > j$ , then we reverse the order of the vertices  $v_{i_j}, \dots, v_{i_k}$  on the corresponding subpath in the current Hamiltonian circuit

$$s : v_1, v_{i_2}, \dots, v_{i_n}, v_1.$$

- If  $k < j$ , then we reverse the order of the vertices  $v_{i_k}, \dots, v_{i_j}$  on the corresponding subpath in the current Hamiltonian circuit

$$s : v_1, v_{i_2}, \dots, v_{i_n}, v_1.$$

We add the missing edges to the graph with very large weights so that we get a complete graph and we will not have to worry about the existence of a Hamiltonian circuit in the first place. If we still do not get a Hamiltonian circuit in the end without those added edges, then there is not a Hamiltonian circuit.

The starting temperature  $T_0$  should be much larger than the values of  $|f(s') - f(s)|$  which guarantees that we can in principle move to any state ("annealing") in the earlier stages of the algorithm. After that, we lower the temperature applying some rule, for example a 10% change.

The annealing algorithm also works for the unsymmetric TSP with obvious changes.

### Karp–Held Heuristics

In the *Karp–Held Heuristics*, we do not directly look for a Hamiltonian circuit but look for a similar subgraph, known as a *spanning 1-tree*<sup>2</sup>. The process does not work for the unsymmetric TSP. The spanning 1-tree  $S_v$  corresponding to the vertex  $v$  (known as the *reference vertex*) is a subgraph of  $G$  that satisfies the following conditions:

- $S_v$  is connected and contains every vertex of  $G$ .
- $S_v$  contains exactly one circuit  $C$  and the vertex  $v$  belongs to  $C$ .
- $S_v$  has exactly two edges incident on  $v$ .

---

<sup>2</sup>Not to be confused with the 1-tree on p. 23!

Clearly, a Hamiltonian circuit is a spanning 1-tree corresponding to any of the vertices. The *weight* of the spanning 1-tree  $S_v$  is the sum of the weights of all its edges, denoted  $\gamma(S_v)$ .  $S_v$  is *minimal* if it has the lowest possible weight.

**Statement.**  $S_v$  is minimal if and only if

- (i)  $S_v - v$  is a minimal spanning tree of  $G - v$ , and
- (ii) the two edges of  $S_v$  incident on  $v$  are the two lightest edges of  $G$  out of all the edges incident on  $v$ .

*Proof.* Let  $S_v$  be a minimal spanning 1-tree. Let  $e$  and  $e'$  be the two edges in  $S_v$  incident on  $v$ . Then,  $S_v - v$  is a spanning tree of  $G - v$  because removing  $v$  destroys the circuit but the connections remain unsevered. If  $S_v - v$  is not a minimal spanning tree of  $G - v$  then there is a lighter spanning tree  $T$  of  $G - v$ . By adding the vertex  $v$  and the edges  $e$  and  $e'$  to  $T$ , we get a spanning 1-tree corresponding to vertex  $v$  which is lighter than  $S_v$  ( $\checkmark$ ). Therefore, (i) is true. Obviously, (ii) is true (because otherwise we would get a lighter spanning 1-tree by replacing  $e$  and  $e'$  with the two lightest edges in  $G$  incident on  $v$ ).

Let us assume that (i) and (ii) are true. If  $S_v$  is not minimal, then there is a lighter minimal spanning 1-tree  $S'_v$  corresponding to  $v$ . Because  $S'_v$  also satisfies (ii), the two edges incident on  $v$  are the same (or at least they have the same weight) in  $S_v$  and  $S'_v$ . Thus,  $S'_v - v$  is lighter than  $S_v - v$  ( $\checkmark$ ).  $\square$

It follows from the statement that any algorithm that finds the minimum spanning tree also works for finding the minimum spanning 1-tree with minor modifications. Especially, Kruskal's and Prim's Algorithms are applicable.

In the Karp–Held Heuristics, we also use weights of vertices, denoted  $\beta(v)$ . With these, we can define the *virtual weight* of an edge as

$$\alpha'(v_i, v_j) = \alpha(v_i, v_j) + \beta(v_i) + \beta(v_j).$$

With the concept of virtual weights, we get the virtual weight of a spanning 1-tree  $S_v$  (we label the edge set of  $S_v$  as  $A$ ):

$$\begin{aligned} \gamma'(S_v) &= \sum_{(v_i, v_j) \in A} \alpha'(v_i, v_j) = \sum_{(v_i, v_j) \in A} \alpha(v_i, v_j) + \sum_{(v_i, v_j) \in A} (\beta(v_i) + \beta(v_j)) \\ &= \gamma(S_v) + \sum_{(v_i, v_j) \in A} (\beta(v_i) + \beta(v_j)). \end{aligned}$$

Now we denote the degree of the vertex  $u$  in  $S_v$  as  $d_{S_v}(u)$ . Then,

$$\sum_{(v_i, v_j) \in A} (\beta(v_i) + \beta(v_j)) = \sum_{i=1}^n \beta(v_i) d_{S_v}(v_i)$$

and

$$\gamma'(S_v) = \gamma(S_v) + \sum_{i=1}^n \beta(v_i) d_{S_v}(v_i).$$

In particular, if we have a Hamiltonian circuit  $H$  (a special spanning 1-tree), then

$$d_H(v_1) = \dots = d_H(v_n) = 2$$

and

$$\gamma'(H) = \gamma(H) + 2 \underbrace{\sum_{i=1}^n \beta(v_i)}_{\text{Does not depend on } H!}.$$

Minimization of the Hamiltonian circuits using virtual weights yields the same minimal circuit than obtained by using real weights. In general though, if we use virtual weights to search for spanning 1-trees, then we get results different from the spanning 1-trees obtained by using real weights.

From now on, we only consider the spanning 1-tree corresponding to vertex  $v_1$  and we leave out the subscript. This is not a limitation of any kind on the Hamiltonian circuits although it might be a good idea to change the reference vertex every now and then. We assume that  $H_{\min}$  is a minimal Hamiltonian circuit and  $S'$  is the minimal spanning 1-tree obtained from using virtual weights (which of course corresponds to  $v_1$ ). Then,

$$\gamma'(H_{\min}) \geq \gamma'(S').$$

In addition,

$$\gamma'(H_{\min}) = \gamma(H_{\min}) + 2 \sum_{i=1}^n \beta(v_i)$$

and

$$\gamma'(S') = \gamma(S') + \sum_{i=1}^n \beta(v_i) d_{S'}(v_i).$$

Thus,

$$\begin{aligned} \gamma(H_{\min}) &= \gamma'(H_{\min}) - 2 \sum_{i=1}^n \beta(v_i) \geq \gamma'(S') - 2 \sum_{i=1}^n \beta(v_i) \\ &= \gamma(S') + \sum_{i=1}^n \beta(v_i) (d_{S'}(v_i) - 2), \end{aligned}$$

from which we get a *lower limit* on  $\gamma(H_{\min})$ .

The idea of the Karp–Held Heuristics is to guide the degrees of the vertices in  $S'$  to the value 2 by changing the weights of the vertices. If we succeed, then we get a minimal Hamiltonian circuit. In all cases, we get a lower limit on the weights  $\gamma(H)$  of the (possible) Hamiltonian circuits by using the calculation above. (Note that  $d_{S'}(v_1)$  is always = 2 if  $S'$  is the spanning 1-tree corresponding to  $v_1$ .)

### The Karp–Held Heuristics:

1. Set  $\beta(v) \leftarrow 0$  for every vertex  $v$ .
2. Set  $\alpha'(u, v) \leftarrow \alpha(u, v) + \beta(u) + \beta(v)$  for each edge  $(u, v)$ .
3. Find the minimal spanning 1-tree  $S'$  using virtual weights  $\alpha'(u, v)$ . If we fail to find this kind of spanning 1-tree, then there is no Hamiltonian circuit and we can stop.
4. If  $S'$  is a circuit, then output the minimal Hamiltonian circuit  $H = S'$  and stop.
5. If  $S'$  is not a circuit and the lower limit calculated from  $S'$  increased during the last  $K$  iterations, then set  $\beta(v) \leftarrow \beta(v) + d_{S'}(v) - 2$  for every vertex  $v$  and go to step #2. ( $K$  is a fixed upper bound on the number of iterations.)

6. If the lower limit calculated from  $S'$  has not increased during the last  $K$  iterations, then output that lower limit and stop.

This procedure does not always produce a minimal Hamiltonian circuit even if there exists one. In practice, it often produces either a minimal Hamiltonian circuit or a good lower limit on the weight of it. Getting a number for the lower limit does not, however, guarantee the existence of a Hamiltonian circuit in the graph!

Karp–Held Heuristics has many steps where we have to choose between different options (such as the reference vertex and the spanning 1-tree). We can not go through every possibility so we must choose randomly. Then, we have a Las Vegas algorithm or stochastic algorithm.

## 5.8 Maximum Matching in Bipartite Graphs: The Hungarian Algorithm

A *matching* in the graph  $G = (V, E)$  is a set of edges  $S \subseteq E$  none of which are adjacent to each other. A matching is a *maximum matching* if it has the greatest possible number of edges. The end vertex of an edge in a matching is *matched*.

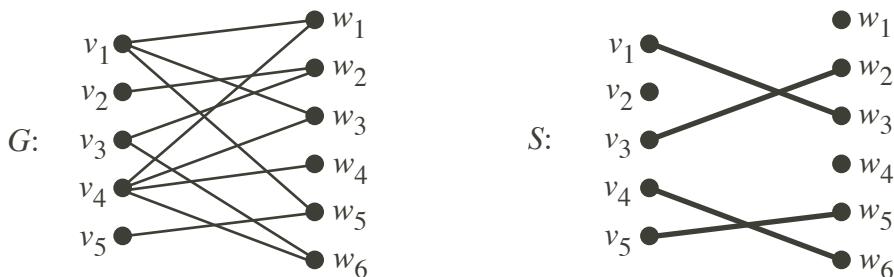
**Problem.** We want to find the maximum matching in a bipartite graph.

An *alternating path* of a matching  $S$  is a path that satisfies the following conditions:

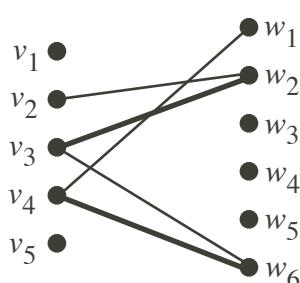
- (1) The first vertex on the path is not matched, and
- (2) every second edge is in the matching and the remaining edges are not in the matching.

Note that the first edge in an alternating path is not in the matching. In addition, if the last vertex of an alternating path is not matched, then this path is an *augmenting path* of  $S$ . A matching without augmenting paths is called a *maximal matching*.

**Example.** For the bipartite graph



one augmenting path of the matching  $S = \{(v_1, w_3), (v_3, w_2), (v_4, w_6), (v_5, w_5)\}$  is the path where the vertices are  $v_2, w_2, v_3, w_6, v_4, w_1$ .

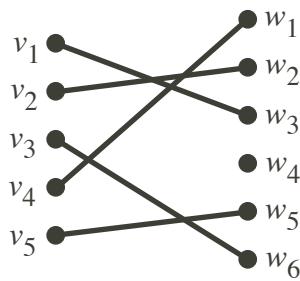


We can *augment* a matching  $S$  using its augmenting path  $p$  as follows:

1. We remove the edges of  $S$  in  $p$ , and
2. We add the edges in  $p$  which are not in  $S$ .

The new edge set is obviously a matching. Note that the number of edges in  $S$  on an augmenting path is one fewer than the number of the remaining edges. Therefore, the number of edges in a matching increases by one after the augmenting operation. It is not possible to augment a maximal matching.

**Example.** (*Continuing from the previous example*) By using the given augmenting path from the matching  $S$ , we get a new maximal matching  $S_1 = \{(v_1, w_3), (v_2, w_2), (v_3, w_6), (v_4, w_1), (v_5, w_5)\}$ .

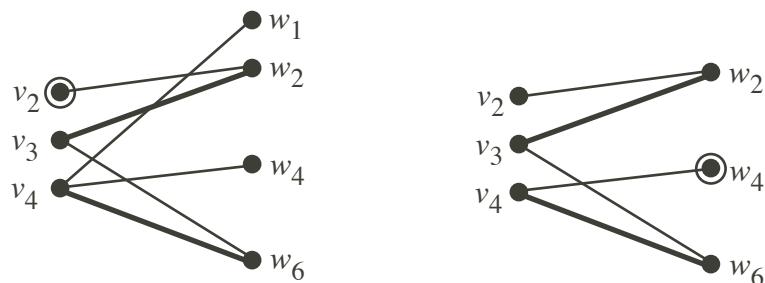


In the *Hungarian Algorithm*, we systematically search for augmenting paths until we get a maximal matching. After that, it suffices to prove that a maximal matching is a maximum matching. From now on, we only consider *bipartite graphs* because the algorithm is then much simpler. We search for augmenting paths by constructing an *alternating tree* of a matching  $S$  which is a subtree of  $G$  such that

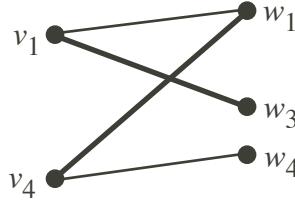
- (1) a vertex  $r$  (the *root* of the tree) is unmatched,
- (2) every second edge on each path out from  $r$  is in  $S$  and the remaining edges are not in  $S$ , and
- (3) either there is an augmenting path out from  $r$  or we can not add any more edges to  $S$ .

An alternating tree is an *augmenting tree* if it has an augmenting path. Otherwise, it is a *Hungarian tree*. Every augmenting path is obviously an augmenting tree by itself. Note that the only unmatched vertex of a Hungarian tree is the root.

**Example.** (*Continuing from the previous example*) Two alternating trees of the matching  $S$  are (the root is circled):



Both of them are augmenting trees. An alternating tree of the matching  $S_1$  (the root is  $w_4$ ) is the Hungarian tree



Augmenting and Hungarian trees are not unique. We can have many different trees depending on the order we take the edges for constructing the trees even though the roots are the same. On the other hand,

**Statement.** *If a matching in a bipartite graph  $G$  has a Hungarian tree, then it does not have an augmenting tree with the same root.*

*Proof.* Let us prove by contradiction and consider the counter hypothesis: A matching  $S$  has a Hungarian tree  $U$  and an augmenting tree  $T$  with the same root  $r$ . We get an augmenting path in the augmenting tree

$$p : r = v_0, e_1, v_1, e_2, \dots, e_k, v_k.$$

We choose the last vertex  $v_i$  which is in  $U$  from the path  $p$  (at least  $r = v_0$  is in  $U$ ). Since  $v_k$  is not in  $U$ ,  $i < k$ . Furthermore, the edge  $e_{i+1}$  is not in the matching nor in  $U$  (otherwise  $v_{i+1}$  would also be in  $U$ ). On the other hand, since  $e_{i+1}$  is not in  $U$ ,  $v_{i+1}$  has to be an end vertex of another edge in  $U$  ( $\checkmark$ ) because the only reason why the edge  $e_{i+1}$  is not put into  $U$  while constructing  $U$  is that the other end vertex  $v_{i+1}$  of  $e_{i+1}$  is already in  $U$ . Note how the bipartiteness of the  $G$  comes in: If the cut in  $G$  that results in the bipartition is  $\langle V_1, V_2 \rangle$ , then the vertices of  $U$  and  $p$  alternate between  $V_1$  and  $V_2$ . Therefore, the length of the  $r-v_i$  path is even in  $p$  and  $U$ .  $\square$

Constructing an alternating tree from a root always leads to a Hungarian tree or an augmenting tree but not both. The order of the edges taken does not matter. (This is not the case for general graphs.)

For the bipartite graph  $G = (V, E)$ , the Hungarian Algorithm is as follows. The cut that yields the bipartition is  $\langle V_1, V_2 \rangle$ .

#### The Hungarian Algorithm:

1. Set  $S \leftarrow \emptyset$ . (We can also use some other initial matching.)
2. If every vertex in  $V_1$  or in  $V_2$  is matched in  $S$ , then  $S$  is a maximum matching and we stop.
3. If there are unmatched vertices in  $S$  of  $V_1$ , then go through them in some order constructing alternating trees (the method of construction is not important as we claimed). If there is an augmenting tree, then augmenting the matching  $S$  by using the augmenting path we have another matching  $S_1$ . Set  $S \leftarrow S_1$  and go to #2.
4. If all the alternating trees that have unmatched vertices in  $V_1$  as roots are Hungarian,  $S$  is maximal and we stop.

**Theorem 5.3.** *A maximal matching in a bipartite graph is a maximum matching.*

*Proof.* Let us prove by contradiction and consider the counter hypothesis: A maximal matching  $S$  in the bipartite graph  $G = (V, E)$  is not a maximum matching. Then, there are more edges in the maximum matching  $S_{\max}$  than in  $S$  and in  $V_1$  there are more vertices matched in  $S_{\max}$  than in  $S$ . We choose an arbitrary vertex  $v \in V_1$ , which is matched in  $S_{\max}$  but not in  $S$ . Then, we have a path

$$p : v = v_0, e_1, v_1, e_2, \dots, e_k, v_k = w,$$

whose edges are alternating between  $S_{\max}$  and  $S$ , i.e.  $e_1$  is in  $S_{\max}$  and  $e_2$  is in  $S$  and so on. We choose the longest such path  $p$ . Because  $p$  is obviously an alternating path of  $S$ , it has even length, i.e.  $e_k$  is an edge of  $S$ . (Otherwise,  $p$  would be an augmenting path of  $S$  which is impossible because  $S$  is maximal.) Thus,  $w$  is matched in  $S$  but not matched in  $S_{\max}$  (because the path  $p$  can not be continued).

Hence, every vertex  $v \in V_1$  which is matched in  $S_{\max}$  but not in  $S$  corresponds to a vertex  $w \in V_1$ , which is matched in  $S$  but not in  $S_{\max}$ . Now, every path that ends at  $w$  must start from the vertex  $v$  if the starting vertex is matched in  $S_{\max}$  but not in  $S$ . The last edge of such a path has to be  $e_k$  (the only edge in  $S$  incident on  $w$ ) and the second to the last vertex has to be  $v_{k-1}$ . Furthermore, the second to the last edge of this path has to be  $e_{k-1}$  (the only edge of  $S_{\max}$  incident on  $v_{k-1}$ ) and the third to the last vertex has to be  $v_{k-2}$ , and so on.

However, there are then in  $V_1$  at least as many vertices  $w$  that are matched in  $S$  but not in  $S_{\max}$  as there are vertices  $v$  that are matched in  $S_{\max}$  but not in  $S$  ( $\checkmark$ ).  $\square$

**Corollary.** *The Hungarian algorithm produces a maximum matching in a bipartite graph.*

A matching is *perfect* if it matches every vertex of a graph. Thus, a graph with an odd number of vertices can not have a perfect matching. Let us consider the graph  $G = (V, E)$  and denote  $\nu(v) = \{\text{adjacent vertices of } v\}$  as well as  $\nu(A) = \bigcup_{v \in A} \nu(v)$  for the vertex set  $A \subseteq V$ . Let us denote by  $\#(X)$  the number of elements in the set  $X$  (the cardinality of the set). With these notions, we can present the following famous characterization:

**Theorem 5.4. (Hall's Theorem or "Marriage Theorem")** *A bipartite graph  $G$  whose bipartition cut is  $\langle V_1, V_2 \rangle$  has a perfect matching if and only if every vertex set  $A \subseteq V_1$  and  $B \subseteq V_2$  satisfies the conditions  $\#(A) \leq \#(\nu(A))$  and  $\#(B) \leq \#(\nu(B))$ .*

*Proof.* If a perfect matching exists, then obviously  $\#(A) \leq \#(\nu(A))$  and  $\#(B) \leq \#(\nu(B))$  hold for all sets of vertices  $A \subseteq V_1$  and  $B \subseteq V_2$ . (Otherwise, we can not find a pair for every vertex in  $A$  or  $B$  in the matching.)

Let us assume that for all sets of vertices  $A \subseteq V_1$  and  $B \subseteq V_2$ ,  $\#(A) \leq \#(\nu(A))$  and  $\#(B) \leq \#(\nu(B))$ . Let  $S$  be a maximum matching in  $G$ . We will prove by contradiction and consider the counter hypothesis:  $S$  is not perfect. We choose a vertex  $v$  which is not matched in  $S$ . Let us examine the case where  $v \in V_1$  (the other case where  $v \in V_2$  is obviously symmetrical). The contradiction is apparent if  $v$  is an isolated vertex so we can move to the case where  $v$  is an end vertex of an edge. The alternating tree with the root  $v$  is then nontrivial and since the matching is also maximal, this tree is Hungarian. We choose such a Hungarian tree  $U$ . We label the set of vertices of  $V_1$  (resp.  $V_2$ ) in  $U$  by  $A$  (resp. by  $B$ ). Because of the construction of  $U$ ,  $B = \nu(A)$ . On the other hand, the vertices of  $A$  and  $B$  in  $U$  are pairwise matched by the edges of  $S$ , except for the root  $r$ . Hence,  $\#(A) = \#(B) + 1 > \#(B)$  ( $\checkmark$ ).  $\square$

## 5.9 Maximum Flow in a Transport Network: The Ford–Fulkerson Algorithm

A *transport network* is a directed graph  $G = (V, E)$  with weighted arcs that satisfies the following:

- (1)  $G$  is connected and loopless,
- (2)  $G$  has only one source  $s$ ,
- (3)  $G$  has only one sink  $t$ , and
- (4) the weight  $c(e)$  of the arc  $e$  is called the *capacity* and it is a nonnegative real number, i.e. we have a mapping  $c : E \rightarrow \mathbb{R}_0$ .

(Compare to stationary linear networks in Section 4.4.) Actually, we could assume that  $G$  has every possible arc except loops and it can even have multiple parallel arcs. If this is not the case, then we simply add the missing arcs with capacity zero. Naturally, we can also assume that  $G$  is nontrivial.

A *flow*  $f$  of a transport network is a weight mapping  $E \rightarrow \mathbb{R}_0$ , which satisfies:

- (i) For each arc  $e$ , we have the *capacity constraint*  $f(e) \leq c(e)$ , and
- (ii) each vertex  $v \neq s, t$  satisfies the *conservation condition* (also called *Kirchhoff's Flow Law*, compare to Section 4.4)

$$\sum_{\substack{\text{initial vertex} \\ \text{of } e \text{ is } v}} f(e) = \sum_{\substack{\text{terminal vertex} \\ \text{of } e \text{ is } v}} f(e).$$

$f(e)$  is called the *flow* of  $e$ . The flow of the arc  $(u, v)$  is also denoted as  $f(u, v)$ . The *value* of the flow  $f$  is

$$|f| = \sum_{\substack{\text{initial vertex} \\ \text{of } e \text{ is } s}} f(e).$$

A flow  $f^*$  is a *maximum flow* if its value is the largest possible, i.e.  $|f^*| \geq |f|$  for every other flow  $f$ .

An  $s$ – $t$  *cut* of a transport network  $S$  is a (directed) cut  $I = \langle V_1, V_2 \rangle$  such that  $s$  is in  $V_1$  and  $t$  is in  $V_2$ . The *capacity* of such a cut is

$$c(I) = \sum_{\substack{u \in V_1 \\ v \in V_2}} c(u, v).$$

(Note that the arcs in the direction opposite to the cut do not affect the capacity.) The capacity of the cut  $\langle V_1, V_2 \rangle$  is also denoted as  $c(V_1, V_2)$ . Furthermore, we define the *flux* of the cut  $I = \langle V_1, V_2 \rangle$  as

$$f^+(I) = \sum_{\substack{u \in V_1 \\ v \in V_2}} f(u, v)$$

and the *counter-flux* as

$$f^-(I) = \sum_{\substack{u \in V_2 \\ v \in V_1}} f(u, v).$$

The value of a flow can now be obtained from the fluxes of any  $s$ – $t$  cut:

**Theorem 5.5.** If  $f$  is a flow of a transport network and  $I$  is an  $s$ - $t$  cut, then

$$|f| = f^+(I) - f^-(I).$$

*Proof.* Obviously,

$$\sum_{\substack{\text{initial vertex} \\ \text{of } e \text{ is } v}} f(e) - \sum_{\substack{\text{terminal vertex} \\ \text{of } e \text{ is } v}} f(e) = \begin{cases} |f| & \text{if } v = s \\ 0 & \text{if } v \neq s, t. \end{cases}$$

We denote  $I = \langle V_1, V_2 \rangle$ . By going through the vertices  $v$  in  $V_1$  and by adding up the equations we get

$$\sum_{\substack{v \in V_1 \\ \text{initial vertex} \\ \text{of } e \text{ is } v}} f(e) - \sum_{\substack{v \in V_1 \\ \text{terminal vertex} \\ \text{of } e \text{ is } v}} f(e) = |f|.$$

For each arc  $e$  whose end vertices are both in  $V_1$ ,  $f(e)$  and  $-f(e)$  are added exactly once and thus they cancel out. Therefore,

$$\sum_{\substack{u \in V_1 \\ v \in V_2}} f(u, v) - \sum_{\substack{u \in V_2 \\ v \in V_1}} f(u, v) = |f|. \quad \square$$

**Corollary.** If  $f$  is a flow of a transport network and  $I$  is an  $s$ - $t$  cut, then  $|f| \leq c(I)$ .

*Proof.*  $|f| = f^+(I) - f^-(I) \leq f^+(I) \leq c(I)$ .  $\square$

An arc  $e$  of a transport network is *saturated* if  $f(e) = c(e)$ . Otherwise, it is *unsaturated*. Now, we point out that  $|f| = c(V_1, V_2)$  if and only if

- (i) the arc  $(u, v)$  is saturated whenever  $u \in V_1$  and  $v \in V_2$ , and
- (ii)  $f(u, v) = 0$  whenever  $u \in V_2$  and  $v \in V_1$ .

An  $s$ - $t$  cut  $I^*$  of a transport network is called a *minimum cut* if  $c(I^*) \leq c(I)$  for every other  $s$ - $t$  cut  $I$ .

**Corollary.** If  $f$  is a flow of a transport network,  $I$  is an  $s$ - $t$  cut and  $|f| = c(I)$ , then  $f$  is a maximum flow and  $I$  is a minimum cut.

*Proof.* If  $f^*$  is a maximum flow and  $I^*$  is a minimum cut, then  $|f^*| \leq c(I^*)$  by the corollary above. Thus,

$$|f| \leq |f^*| \leq c(I^*) \leq c(I)$$

and  $f$  is indeed a maximum flow and  $I$  is indeed a minimum cut.  $\square$

Actually, the value of the maximum flow is the capacity of the minimum cut. To show this, we examine a path from vertex  $s$  to vertex  $v$  (not necessarily a directed path):

$$s = v_0, e_1, v_1, e_2, \dots, e_k, v_k = v \quad (\text{path } p).$$

If  $e_i = (v_{i-1}, v_i)$ , then the arc  $e_i$  is a *forward arc*. If  $e_i = (v_i, v_{i-1})$ , then the arc  $e_i$  is a *back arc*. The arc  $e_i$  of  $p$  is now weighted by the following formula:

$$\epsilon(e_i) = \begin{cases} c(e_i) - f(e_i) & \text{if } e_i \text{ is a forward arc} \\ f(e_i) & \text{if } e_i \text{ is a back arc} \end{cases}$$

and the path  $p$  is weighted by the following formula:

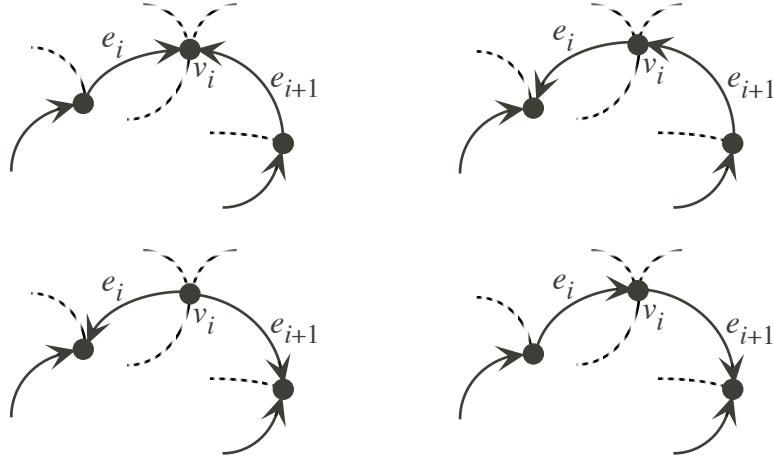
$$\epsilon(p) = \min_{i=1}^k \{\epsilon(e_i)\}.$$

The path  $p$  is *unsaturated* if  $\epsilon(p) > 0$ , i.e. all of the forward arcs of  $p$  are unsaturated and  $f(e_i) > 0$  for all the back arcs  $e_i$  of  $p$ .

In particular, an  $s-t$  path can be unsaturated in which case it is called an *augmenting path*<sup>3</sup>. All of these definitions are of course attached to a certain flow  $f$ . By starting from an  $s-t$  path  $p$  (and a flow  $f$ ), we can define a new flow:

$$\bar{f} = \begin{cases} f(e) + \epsilon(p) & \text{if } e \text{ is a forward arc of } p \\ f(e) - \epsilon(p) & \text{if } e \text{ is a back arc of } p \\ f(e) & \text{otherwise.} \end{cases}$$

$\bar{f}$  is really a flow. Changes in  $f$  can only occur at the arcs and vertices of  $p$ . Every arc of  $p$  satisfies the capacity constraint because of how  $\epsilon(p)$  and  $\bar{f}(e)$  are defined. A vertex  $v_i$  of  $p$  satisfies the conservation condition which we can verify. We have four cases:



Obviously (think about the source  $s$ )

$$|\bar{f}| = |f| + \epsilon(p),$$

so  $f$  is not a maximum flow if it has an augmenting path. Moreover, the converse is true as well. Hence,

**Theorem 5.6.** *A flow is a maximum flow if and only if it does not have any augmenting path.*

*Proof.* As we claimed, a maximum flow can not have an augmenting path. Let us assume that a flow  $f$  does not have an augmenting path. We denote the set of vertices which we can reach from the source  $s$  along unsaturated paths by  $V_1$ . Then, trivially,  $s \in V_1$  and  $t \notin V_1$  (because there are no augmenting paths). Thus, the cut  $I = \langle V_1, V_2 \rangle$  is an  $s-t$  cut. We proceed to prove that  $|f| = c(I)$ . By the previous corollary,  $f$  is then a maximum flow.

Let us consider the arc  $(u, v)$ , where  $u \in V_1$  and  $v \in V_2$ . Then, there exists an unsaturated  $s-u$  path  $p$ . The edge  $(u, v)$  is saturated because there would be an unsaturated  $s-v$  path otherwise. Similarly, we conclude that  $f(u, v) = 0$  for every arc  $(u, v)$ , where  $u \in V_2$  and  $v \in V_1$ . Therefore, the flux  $f^+(I)$  is  $c(I)$  and the counter-flux  $f^-(I)$  is zero. By Theorem 5.5,  $|f| = c(I)$ .  $\square$

<sup>3</sup>Not to be confused with the augmenting path in the previous section!

We have also proven the celebrated

**Theorem 5.7. (Max-Flow Min-Cut Theorem)** *The value of a maximum flow in a transport network is the same as the capacity of a minimum cut.*

If the capacities of the arcs are rational numbers, then a maximum flow can be found by using Theorem 5.6. The algorithm tries to find an augmenting path for  $f$ . If it can not be found, then we have a maximum flow. If we find an augmenting path, then we use it to create a greater flow  $\bar{f}$ . In the algorithm, we use a label  $\alpha$  for the vertices in the following way:

$$\alpha(v) = (u, \text{direction}, \Delta),$$

where  $u$  is a vertex in the transport network (or – if it is not defined), "direction" is either forward ( $\rightarrow$ ) or back ( $\leftarrow$ ) (or – if it is not defined) and  $\Delta$  is a nonnegative real number (or  $\infty$ ). The point is, whenever a vertex  $v$  is labeled, there is an  $s-v$  path  $p$  which contains the ("directed") arc  $(u, v)$  and  $\Delta = \epsilon(p)$ . A direction is forward if an arc is in the direction of the path and back otherwise. We can label a vertex  $v$  when the vertex  $u$  has been labeled and either  $(u, v)$  or  $(v, u)$  is an arc. We have two cases:

- (1) (*Forward Label*) If  $e = (u, v)$  is an arc and  $\alpha(u) = (\cdot, \cdot, \Delta_u)$  as well as  $c(e) > f(e)$ , then we can write  $\alpha(v) = (u, \rightarrow, \Delta_v)$ , where

$$\Delta_v = \min\{\Delta_u, c(e) - f(e)\}.$$

- (2) (*Back Label*) If  $e = (v, u)$  is an arc and  $\alpha(u) = (\cdot, \cdot, \Delta_u)$  as well as  $f(e) > 0$ , then we can write  $\alpha(v) = (u, \leftarrow, \Delta_v)$ , where

$$\Delta_v = \min\{\Delta_u, f(e)\}.$$

There are two phases in the algorithm. In the first phase, we label the vertices as presented above and each vertex is labeled at most once. The phase ends when the sink  $t$  gets labeled as  $\alpha(t) = (\cdot, \rightarrow, \Delta_t)$ , or when we can not label any more vertices. In the second case, there are no augmenting paths and the flow we obtain is a maximum flow so we stop. In the first case, the flow we obtain is not a maximum flow and we have an augmenting path  $p$  for which  $\epsilon(p) = \Delta_t$ . The algorithm moves on to the second phase. In the second phase, we construct a new greater flow  $\bar{f}$  by using the labels of the vertices of  $p$  obtained previously. After this, we go back to the first phase with this greater new flow.

### The Ford–Fulkerson Algorithm:

1. Choose an initial flow  $f_0$ . If we do not have a specific flow in mind, we may use  $f_0(e) = 0$ . Label the source  $s$  by  $\alpha(s) \leftarrow (-, -, \infty)$ . Set  $f \leftarrow f_0$ .
2. If we have an unlabeled vertex  $v$ , which can be labeled either forward by  $(w, \rightarrow, \Delta_v)$  or backward by  $(w, \leftarrow, \Delta_v)$ , then we choose one such vertex and label it. (There can be many ways of doing this and all of them are permitted.) If such a vertex  $v$  does not exist, output the maximum flow  $f$  and stop.
3. If  $t$  has not been labeled, go to step #2. Otherwise, set  $u \leftarrow t$ .

4. If  $\alpha(u) = (w, \rightarrow, \Delta_u)$ , then set

$$f(w, u) \leftarrow f(w, u) + \Delta_t \quad \text{and} \quad u \leftarrow w.$$

If  $\alpha(u) = (w, \leftarrow, \Delta_u)$ , then set

$$f(u, w) \leftarrow f(u, w) - \Delta_t \quad \text{and} \quad u \leftarrow w.$$

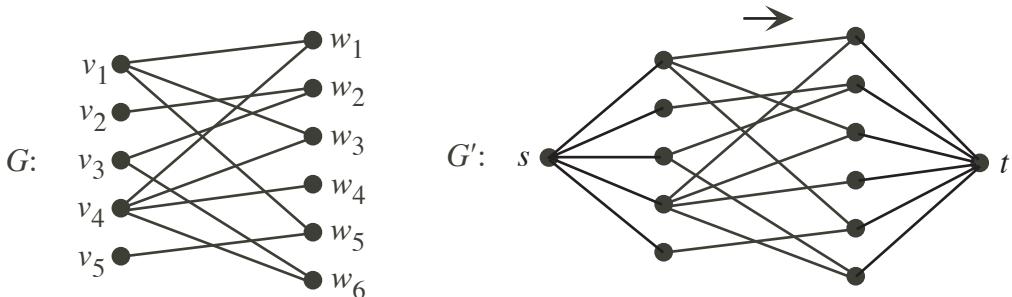
5. If  $u = s$ , then remove all the labels  $\alpha$  but not the label of the source and go to step #2. If  $u \neq s$ , then go to step #4.

If  $f_0(e)$  and  $c(e)$  are rational numbers, then the algorithm stops and produces a maximum flow.<sup>4</sup> In this case, we can assume that these weights and capacities are nonnegative integers. Thus, the value of a flow increases by a positive integer every time we move from the second phase to the first phase and the value reaches a maximum eventually. On the other hand, the number of steps can be as large as the value of the maximum flow. The performance time of the algorithm does not only depend on the number of vertices but also the capacities.

The algorithm can be modified<sup>5</sup> so that it does not depend on the capacities. Thus, it will work for irrational capacities. In this case, our purpose during the labeling phase is to find the shortest augmenting path. We get this by always choosing the vertex  $v$  in step #2 in such a way that in  $\alpha(v) = (w, \cdot, \Delta_v)$ ,  $w$  received its label as early as possible.

The Ford–Fulkerson Algorithm also works for finding a maximum matching in a bipartite graph. Let us do an example:

**Example.** Using the bipartite graph  $G$  from an example in the previous section, we get a transport network  $G'$ :



Every edge of  $G'$  is directed from left to right and given a capacity of 1. The initial flow is a zero flow (or a greater flow we obtain from some other initial flow). During the whole process, the flows of the edges are integers 0 or 1. We take into the matching those edges in  $G$  whose corresponding edges  $e$  in  $G'$  receive a flow  $f(e) = 1$  and a maximum flow gives a maximum matching. Note that an augmenting path can be of length larger than three in this case. (We can also claim now that the augmenting paths here and the augmenting paths obtained from the Hungarian Algorithm do have something in common after all!)

<sup>4</sup>If there are irrational capacities or flows  $f_0(e)$ , then the algorithm may not stop at all and it may not produce a maximum flow even if the process repeats endlessly. Of course, we do not have to use irrational flows. In practice, we will not use irrational capacities.

<sup>5</sup>This is known as the *Edmonds–Karp Modification* (refer e.g. to SWAMY & THULASIRAMAN).

# Chapter 6

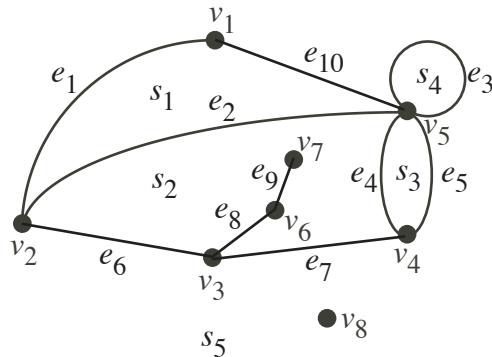
## Drawing Graphs

### 6.1 Planarity and Planar Embedding

We have not treated graphs as geometric objects so far in the course. In practice, we *draw* graphs, i.e. we treat vertices as geometric points and edges as continuous curves. If a graph  $G$  can be drawn on a plane (or a sphere) so that the edges only intersect at vertices, then it is *planar*. Such a drawing of a planar graph is a *planar embedding* of the graph.

A connected part of a plane which does not contain any vertices and is surrounded by edges is called a *region* of a planar embedding. In addition, the part outside the embedding is considered as a region, known as the *exterior region* (when we draw a planar graph on a plane or on a sphere, it is just like any other region). The vertices surrounding a region  $s$  are called *boundary vertices* and the edges surrounding  $s$  are called *boundary edges*. Two regions are *adjacent* if they share a boundary edge. Note that a region can be adjacent to itself.

**Example.** In the following planar embedding



the regions are  $s_1$ ,  $s_2$ ,  $s_3$ ,  $s_4$  and  $s_5$  (the exterior region) and their boundary vertices and edges as well as their adjacent regions are given in the table below:

region	boundary vertices	boundary edges	adjacent regions
$s_1$	$v_1, v_5, v_2$	$e_1, e_{10}, e_2$	$s_2, s_5$
$s_2$	$v_2, v_5, v_4, v_3, v_6, v_7$	$e_2, e_4, e_7, e_9, e_8, e_6$	$s_1, s_2, s_3, s_5$
$s_3$	$v_4, v_5$	$e_4, e_5$	$s_2, s_5$
$s_4$	$v_5$	$e_3$	$s_5$
$s_5$	$v_1, v_5, v_4, v_3, v_2, v_8$	$e_{10}, e_3, e_5, e_7, e_6, e_1$	$s_1, s_2, s_3, s_4$

In the following, we investigate some fundamental properties of planar embeddings of graphs.

**Theorem 6.1. (Euler's Polyhedron Formula<sup>1</sup>)** *If a planar embedding of a connected graph  $G$  has  $n$  vertices,  $m$  edges and  $f$  regions, then*

$$f + n = m + 2.$$

*Proof.* Let us use induction on  $m$ .

Induction Basis:  $m = 0$ . Planar embedding of  $G$  has only one vertex and one region (the exterior region) so the claim is true.

Induction Hypothesis: The theorem is true for  $m \leq \ell$ . ( $\ell \geq 0$ )

Induction Statement: The theorem is true for  $m = \ell + 1$ .

Induction Statement Proof: We choose an edge  $e$  of  $G$  and examine the graph  $G' = G - e$ . If  $e$  is in a circuit, then  $G'$  is connected and by the Induction Hypothesis, we get

$$f' + n = (m - 1) + 2,$$

where  $f'$  is the number of regions in  $G'$ . However, closing the circuit with  $e$  increases the number of regions by one so  $f' = f - 1$  and the theorem is true. If  $G - e$  is disconnected, then it has two planar components,  $G_1$  and  $G_2$  whose number of vertices, edges and regions are  $n_1, n_2, m_1, m_2, f_1$  and  $f_2$ , respectively. By the Induction Hypothesis,

$$f_1 + n_1 = m_1 + 2 \quad \text{and} \quad f_2 + n_2 = m_2 + 2.$$

While adding  $e$ , the number of regions becomes  $f_1 + f_2 - 1$  ( $G_1$  and  $G_2$  share the same exterior region or one exterior region is drawn to be a region of the other component), the number of vertices becomes  $n_1 + n_2$  and the number of edges becomes  $m_1 + m_2 + 1$ . Hence, the claim is true.  $\square$

**Example.** (Continuing from the previous example) We remove the vertex  $v_8$  to get a connected planar embedding. Now, we have 7 vertices, 10 edges, 5 regions and  $5 + 7 = 10 + 2$ .

**Theorem 6.2. (The Linear Bound)** *If a simple connected planar graph  $G$  has  $n \geq 3$  vertices and  $m$  edges, then*

$$m \leq 3n - 6.$$

*Proof.* If the regions of a planar embedding of  $G$  are  $s_1, \dots, s_f$ , then we denote the number of boundary edges of  $s_i$  by  $r_i$  ( $i = 1, \dots, f$ ). The case  $f = 1$  is obvious because  $G$  is then a tree and  $m = n - 1 \leq 3n - 6$ . Thus, we assume that  $f \geq 2$ . Since  $G$  is simple, every region has at least 3 boundary edges and thus

$$\sum_{i=1}^f r_i \geq 3f.$$

Every edge is a boundary edge of one or two regions in the planar embedding, so

$$\sum_{i=1}^f r_i \leq 2m.$$

The result now follows directly from Euler's Polyhedron Formula.  $\square$

---

<sup>1</sup>The name comes from a polyhedron with  $n$  vertices,  $m$  edges,  $f$  faces and no holes.

**Theorem 6.3. (The Minimum Degree Bound)** For a simple planar graph  $G$ ,  $\delta(G) \leq 5$ .

*Proof.* Let us prove by contradiction and consider the counter hypothesis:  $G$  is a simple planar graph and  $\delta(G) \geq 6$ . Then, (by Theorem 1.1)  $m \geq 3n$ , where  $n$  is the number of vertices and  $m$  is the number of edges in  $G$ . ( $\checkmark$  Theorem 6.2)  $\square$

A characterization of planar graphs is obtained by examining certain forbidden subgraphs.

**Theorem 6.4. (Kuratowski's Theorem)** A graph is planar if and only if none of its subgraphs can be transformed to  $K_5$  or  $K_{3,3}$  by contracting edges.

The proof is quite complicated (but elegant!), refer e.g. to SWAMY & THULASIRAMAN for more information.  $K_5$  and  $K_{3,3}$  are not planar, which can be verified easily.

There are many fast but complicated algorithms for testing planarity and drawing planar embeddings. For example, the *Hopcroft–Tarjan Algorithm*<sup>2</sup> is one. We present a slower classical polynomial time algorithm, the *Demoucron–Malgrange–Pertuiset Algorithm*<sup>3</sup> (usually just called *Demoucron's Algorithm*). The idea of the algorithm is to try to draw a graph on a plane piece by piece. If this fails, then the graph is not planar.

If  $G$  is a graph and  $R$  is a planar embedding of a planar subgraph  $S$  of  $G$ , then an *R-piece*  $P$  of  $G$  is

- either an edge of  $G - S$  whose end vertices are in  $S$ , or
- a component of the subgraph induced by vertices not in  $S$  which contains the edges (if any) that connect  $S$  to the component, known as *pending edges*, and their end vertices.

Those vertices of an *R-piece* of  $G$  that are end vertices of pending edges connecting them to  $S$  are called *contact vertices*. We say that a planar embedding  $R$  of the planar subgraph  $S$  is *planar extendable* to  $G$  if  $R$  can be extended to a planar embedding of the whole  $G$  by drawing more vertices and/or edges. Such an extended embedding is called a *planar extension* of  $R$  to  $G$ . We say further that an *R-piece*  $P$  of  $G$  is *drawable* in a region  $s$  of  $R$  if there is a planar extension of  $R$  to  $G$  where  $P$  is inside  $s$ . Obviously all contact vertices of  $P$  must then be boundary vertices of  $s$ , but this is of course not sufficient to guarantee planar extendability of  $R$  to  $G$ . Therefore we say that a  $P$  is *potentially drawable* in  $s$  if its contact vertices are boundary vertices of  $s$ . In particular, a piece with no contact vertices is potentially drawable in any region of  $R$ .

### Demoucron's Algorithm:

1. We first check whether or not  $G$  is a forest. If it is a forest, then it clearly is planar and can be planar embedded. (There are fast algorithms for this purpose.) We can then stop.
2. If  $G$  is not a forest then it must contain at least one circuit. We choose a circuit  $C$ , embed it to get the planar embedding  $D$ , and set  $R \leftarrow D$ . (A circuit is obviously planar and is easily planar embedded.)
3. If  $R$  is a planar embedding of  $G$ , then we output it and stop.

<sup>2</sup>The original reference is HOPCROFT, J.E. & TARJAN, R.E.: Efficient Planarity Testing. *Journal of the ACM* **21** (1974), 549–568.

<sup>3</sup>The original reference is DEMOUCRON, G. & MALGRANGE, Y. & PERTUISET, R.: Graphes planaires: reconnaissance et construction des représentations planaires topologiques. *Revue Française Recherche Opérationnelle* **8** (1964), 33–47.

4. We construct the set  $\mathcal{P}$  of all  $R$ -pieces of  $G$ . For each piece  $P \in \mathcal{P}$  we denote by  $\mathcal{S}(P)$  the set of all those regions of  $R$  which  $P$  is potentially drawable in.
5. If, for an  $R$ -piece  $P \in \mathcal{P}$ , the set  $\mathcal{S}(P)$  is empty then  $G$  is not planar. We can then output this information and stop.
6. Choose an  $R$ -piece  $P$ , starting from those potentially drawable only in one region.
7. Depending on the number of contact vertices of  $P$ , we planar extend  $R$ :
  - 7.1 If  $P$  has no contact vertices, we call Demoucron's Algorithm recursively with input  $P$ . If it turns out that  $P$  is not planar, then  $G$  is not planar, and we output this information and stop. Otherwise we extend  $R$  to a planar embedding  $U$  by drawing  $P$  in one of its regions, set  $R \leftarrow U$ , and return to step #3.
  - 7.2 If  $P$  has exactly one contact vertex  $v$ , with the corresponding pendant edge  $e$ , we call Demoucron's Algorithm recursively with input  $P$ . If it turns out that  $P$  is not planar, then  $G$  is not planar, and we output this information and stop. Otherwise we extend  $R$  to a planar embedding  $U$  by drawing  $P$  in a region with boundary vertex  $v$ , set  $R \leftarrow U$ , and return to step #3. (This region of  $R$  will then be an exterior region of the planar embedding of  $P$ .)
  - 7.3 If  $P$  has (at least) two contact vertices  $v_1$  and  $v_2$ , they are connected by a path  $p$  in  $P$ . We then extend  $R$  to a planar embedding  $U$  by drawing  $p$  in a region of  $R$  with boundary vertices  $v_1$  and  $v_2$  where  $P$  is potentially drawable, set  $R \leftarrow U$ , and return to step #3.

Clearly, if  $G$  is not planar, Demoucron's Algorithm will output this information. On the other hand, the algorithm will not get stuck without drawing the planar embedding if the input is planar, because

**Statement.** *If  $G$  is planar, then at each step of the algorithm  $R$  is planar extendable to  $G$ .*

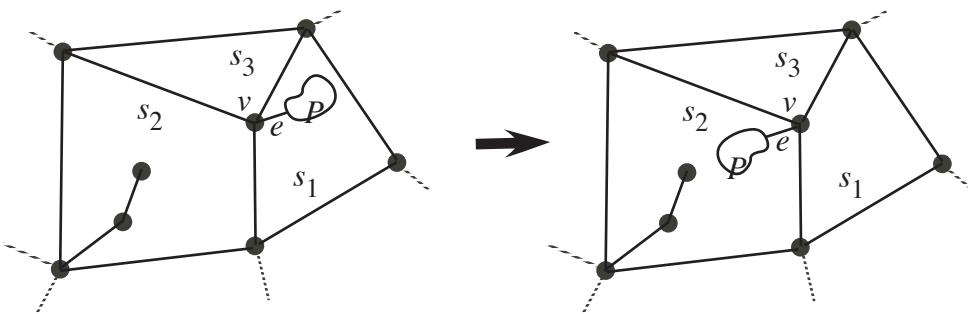
*Proof.* We use induction on the number of times  $\ell$  the algorithm visits step #7.

Induction Basis:  $\ell = 0$ . Now either  $G$  is a forest (and  $R$  is not needed) or  $R$  is a circuit of  $G$ . Obviously the planar embedding of this circuit can be planar extended to  $G$ .

Induction Hypothesis: The statement is true for  $\ell \leq r$ . ( $r \geq 0$ )

Induction Statement: The statement is true for  $\ell = r + 1$ .

Induction Statement Proof: For step #7.1 the matter is clear. If, in step #7.2,  $P$  is potentially drawable in the region  $s$  of  $R$ , it can always be drawn in this region without endangering subsequent steps. In other words, any possible region can be chosen. This is because the region can be exchanged for another at all times by "reflection" with respect to the vertex  $v$  (and possibly rescaling):



Similarly, if in step #7.3,  $P$  is drawable in a region of  $R$ , then it can be drawn in this region without endangering subsequent steps. If  $P$  is drawable in both region  $s_1$  and region  $s_2$ , its contact vertices are boundary vertices of both  $s_1$  and  $s_2$ . At any time, a drawn  $P$  (or part of it) can be moved from region  $s_1$  to  $s_2$ , or vice versa, simply by reflection with respect to the common boundary (and possibly rescaling to fit into the region).  $\square$

**Remark.** Nonplanar graphs may be embedded on closed continuous surfaces with holes. For instance, a torus is closed surface with exactly one hole. On a torus we can embed the non-planar graphs  $K_5$ ,  $K_6$  and  $K_7$ , and also  $K_{3,3}$ .  $K_8$  is more complex and its embedding requires a closed surface with two holes. The smallest number of holes in a closed surface required for embedding the graph  $G$  on it is called the genus of  $G$ . On the other hand the smallest number of crossings of edges in a drawing of  $G$  on plane is called the crossing number of  $G$ . Computation of genus and crossing number are both  $\mathcal{NP}$ -complete problems.

A *coloring* of a graph is a labeling of vertices where adjacent vertices never share a label. The labels are then often called *colors*. We say that a graph is  $k$ -colorable if it can be colored using (at most)  $k$  colors. If a graph is colorable then it obviously can not have loops. Equally obviously, parallel edges can be reduced to one, so we may assume our graphs here to be simple. The smallest number  $k$  for which the graph  $G$  is  $k$ -colorable, is called the *chromatic number* of  $G$ , denoted by  $\chi(G)$ .

$K_4$  is an example of a planar simple graph which is not 3-colorable. On the other hand there is the celebrated

**Theorem 6.5. (The Four-Color Theorem)** Every simple planar graph is 4-colorable.

*Proof.* The only known proofs require extensive computer runs. The first such proof was obtained by Kenneth Appel ja Wolfgang Haken in 1976. It takes a whole book to present the proof: APPEL, K. & HAKEN, W.: *Every Planar Map is Four Colorable*. American Mathematical Society (1989).  $\square$

If we require a bit less, i.e. 5-colorability, then there is much more easily provable result, and an algorithm.

**Theorem 6.6. (Heawood's Theorem or The Five-Color Theorem)** Every simple planar graph is 5-colorable.

*Proof.* We may think of  $G$  as a planar embedding. We use induction on the number  $n$  of vertices of  $G$ .

Induction Basis:  $n = 1$ . Our graph is now 1-colorable since there are no edges.

Induction Hypothesis: The theorem is true for  $n \leq \ell$ . ( $\ell \geq 1$ )

Induction Statement: The theorem is true for  $n = \ell + 1$ .

Induction Statement Proof: According to the Minimum Degree Bound, there is a vertex  $v$  in  $G$  of degree at most 5. On the other hand, according to the Induction Hypothesis the graph  $G - v$  is 5-colorable. If, in this coloring, the vertices adjacent to  $v$  are colored using at most four colors, then clearly we can 5-color  $G$ .

So we are left with the case where the vertices  $v_1, v_2, v_3, v_4, v_5$  adjacent to  $v$  are colored using different colors. We may assume that the indexing of the vertices proceeds clockwise, and we label the colors with the numbers 1, 2, 3, 4, 5 (in this order). We show that the coloring of  $G - v$  can be changed so that (at most) four colors suffice for coloring  $v_1, v_2, v_3, v_4, v_5$ .

We denote by  $H_{i,j}$  the subgraph of  $G - v$  induced by the vertices colored with  $i$  and  $j$ . We have two cases:

- $v_1$  and  $v_3$  are in different components  $H_1$  and  $H_3$  of  $H_{1,3}$ . We then interchange the colors 1 and 3 in the vertices of  $H_3$  leaving the other colors untouched. In the resulting 5-coloring of  $G - v$  the vertices  $v_1$  and  $v_3$  both have the color 1. We can then give the color 3 to  $v$ .
- $v_1$  and  $v_3$  are connected in  $H_{1,3}$ . Then there is a  $v_1-v_3$  path in  $H_{1,3}$ . Including the vertex  $v$  we get from this path a circuit  $C$ . Now, since we indexed the vertices  $v_1, v_2, v_3, v_4, v_5$  clockwise, exactly one of the vertices  $v_2$  and  $v_4$  is inside  $C$ . We deduce that  $v_2$  and  $v_4$  are in different components of  $H_{2,4}$ , and we have a case similar to the previous one.  $\square$

The proof gives a simple (recursive) algorithm for 5-coloring a planar graph, the so-called *Heawood's Algorithm*.

## 6.2 The Davidson–Harel Algorithm

For the actual drawing of a graph we need to define the drawing area (the "window"), i.e. a rectangular area with sides parallel to the coordinate axes, the drawing curve of the edges (here edges are drawn as line segments), and certain "criteria of beauty", so that the resulting drawing is pleasant to the eye, balanced, and as clear as possible. Such "beauty criteria" are of course context-dependent and even matters of individual taste. In the sequel we restrict ourselves to simple graphs, given by, say, an adjacency matrix or an all-vertex incidence matrix.

We will now present the so-called *Davidson–Harel Algorithm*<sup>4</sup> which, applying an annealing algorithm, aims at better and better drawings of a graph using a certain *ugliness function* (cf. Section 5.7). An ugliness function  $R$  computes a numerical *ugliness value* obtained from a drawing  $P$  of a graph  $G$ . This value is a sum of various contributing factors. We denote, as usual, the sets of vertices and edges of  $G$  by  $\{v_1, \dots, v_n\}$  and  $\{e_1, \dots, e_m\}$ , respectively. We also denote by  $\mathbf{v}_i$  the vector (or geometric point) corresponding to the vertex  $v_i$ , and by  $\mathbf{e}_j$  the line segment corresponding to the edge  $e_j$ . Further, we denote

$$d_{ij} = \|\mathbf{v}_i - \mathbf{v}_j\|,$$

$r_i$  = distance of  $\mathbf{v}_i$  from the right border of the window,

$l_i$  = distance of  $\mathbf{v}_i$  from the left border of the window,

$u_i$  = distance of  $\mathbf{v}_i$  from the upper border of the window,

$b_i$  = distance of  $\mathbf{v}_i$  from the lower border of the window,

$c_j$  = length of the line segment  $\mathbf{e}_j$ ,

$$f_{ij} = \begin{cases} 1, & \text{if the line segments } \mathbf{e}_i \text{ and } \mathbf{e}_j \text{ intersect without } e_i \text{ and } e_j \text{ being adjacent} \\ 0 & \text{otherwise,} \end{cases}$$

$$g_{ij} = \begin{cases} \text{distance of } \mathbf{v}_i \text{ from the line segment } \mathbf{e}_j \text{ if it exceeds } \gamma \text{ and } v_i \text{ is not an end vertex of } e_j \\ \gamma \text{ otherwise.} \end{cases}$$

$\gamma$  is a parameter of the algorithm telling how close to vertices edges can be. The ugliness function is then given by

---

<sup>4</sup>The original reference is DAVIDSON, R. & HAREL, D.: Drawing Graphs Nicely Using Simulated Annealing. *ACM Transactions on Graphics* **15** (1996), 301–331.

$$\begin{aligned}
R(P) = & \lambda_1 \sum_{i=1}^{n-1} \sum_{j=i+1}^n \frac{1}{d_{ij}^2} + \lambda_2 \sum_{i=1}^n \left( \frac{1}{r_i^2} + \frac{1}{l_i^2} + \frac{1}{u_i^2} + \frac{1}{b_i^2} \right) + \lambda_3 \sum_{j=1}^m c_j^2 \\
& + \lambda_4 \sum_{i=1}^{m-1} \sum_{j=i+1}^m f_{ij} + \lambda_5 \sum_{i=1}^n \sum_{j=1}^m \frac{1}{g_{ij}^2},
\end{aligned}$$

where  $\lambda_1, \dots, \lambda_5$  are nonnegative-valued parameters weighting the contributions of the various factors. (One could actually use negative values as well, whatever the interpretation then might be.)

We can compute  $d_{ij}, \dots, g_{ij}$  quite easily using some basic formulae of vector geometry. We must, however, think about the speed of the computation as well. One way to speed up the computation is to use complex arithmetic.  $d_{ij}, \dots, g_{ij}$  are then equally easily computable.<sup>5</sup> It may also be of advantage to force the vertices into a lattice of geometric points. This can be achieved for instance by rounding the coordinates (or complex numbers) to a fixed accuracy and abandoning drawings where the ugliness function has the value  $\infty$  (this happens e.g. when vertices occupy the same point).

In the annealing process the state is  $P$  and the response is  $R(P)$ . An initial state can be obtained by choosing the points  $\mathbf{v}_1, \dots, \mathbf{v}_n$  in the window randomly, and then drawing the edges accordingly. The state transition process  $P \leftarrow A_\rho(P)$  is the following:

- Choose a random vertex  $\mathbf{v}_i$ . (Alternatively the vertices may be circulated cyclically.)
- Draw a circle of radius  $\rho$  centered on  $\mathbf{v}_i$ . The radius  $\rho$  is a parameter, which is initially large and gradually reduced later in some systematic fashion.
- Choose a random point  $\mathbf{u}$  on this circle.
- If  $\mathbf{u}$  is outside the drawing window, the state remains the same. Otherwise set  $\mathbf{v}_i \leftarrow \mathbf{u}$  and change the edges accordingly in the drawing.

The remaining parts of the algorithm are very similar to the annealing algorithm for the TSP in Section 5.7.

**Remark.** *This method has numerous variants. The window could be a circle and the edges concentric arcs or radii. Or the window could be a sphere and edges drawn as arcs of great circles. The window could also be unbounded, for instance, the whole of  $\mathbb{R}^2$ . We could "draw" graphs three-dimensionally. Etc. We could also use a metric other than the Euclidean one when computing distances, e.g. the Manhattan metric ("1-norm") or the max-metric (" $\infty$ -norm"), geodetic distances on a sphere, etc. Needless to say, the resulting drawings are rather different using these variants of the algorithm.*

*It may be noted that using nearly any effective criteria, finding the optimally pleasing drawing of a simple graph is an  $\mathcal{NP}$ -hard problem.*

---

<sup>5</sup>Note that if  $z_1 = x_1 + jy_1$  and  $z_2 = x_2 + jy_2$ , where  $j$  is the imaginary unit, then the real part of  $\overline{z_1}z_2$  equals the dot product  $(x_1, y_1) \bullet (x_2, y_2)$  and the imaginary part equals the determinant  $\begin{vmatrix} x_1 & x_2 \\ y_1 & y_2 \end{vmatrix}$ .

# Chapter 7

## MATROIDS

Many concepts in the preceding chapters do not so much deal with graphs themselves as their structural properties. Examples are various dualities (cut set vs. circuit), principles behind certain algorithms (e.g. Kruskal's Algorithms), and various extremality properties (many structures are the "smallest of their kind", one cannot e.g. remove an edge of a cut set without it losing this property).

Exactly corresponding structures were found in many other areas of mathematics, and they were called matroids.<sup>1</sup>

### 7.1 Hereditary Systems

A *hereditary family of sets* is a family of sets such that whenever a set  $F$  is in the family then so are all subsets of  $F$  (and in particular the empty set  $\emptyset$ ). A *hereditary system*  $M$  of a set  $E$  is a nonempty hereditary family  $\mathcal{I}_M$  of subsets of  $E$ . Included there are also the various ways of specifying  $\mathcal{I}_M$ , called *aspects*. It will be assumed in what follows that  $E$  is a finite set. The following nomenclature is traditional:

- Sets in the family  $\mathcal{I}_M$  are called *independent sets* of  $M$ .
- The family of subsets of  $E$  other than those in  $\mathcal{I}_M$  is denoted by  $\mathcal{D}_M$  and called the family of *dependent sets* of  $M$ .
- An independent set is *maximal* if it is not a proper subset of another independent set. A maximal independent set is called a *basis*. The family of all bases is denoted by  $\mathcal{B}_M$ . Note that an independent set is always contained in a basis.
- A dependent set is *minimal* if no dependent set is its proper subset. A minimal dependent set is called a *circuit*.<sup>2</sup> (Recall that the empty set is always in  $\mathcal{I}_M$ .) The family of all circuits is denoted by  $\mathcal{C}_M$ . Note that a dependent set always contains a circuit.
- A circuit consisting of only one element is a so-called *loop*. Elements of a circuit with two elements are called *parallel*. A hereditary system is *simple* if it has no loops and no parallel elements.

---

<sup>1</sup>The remarkable thing is that many of these structures were found independently at the same time around the year 1935: Hassler Whitney investigated planarity of graphs, Saunders MacLane geometric lattices of points, and Bartel van der Waerden's topic was independence in vector spaces.

<sup>2</sup>This or any other "familiar sounding" concept should not be confused with the corresponding concept for graphs, even though there is a certain connection, as will be seen!

- The *rank* of a subset  $F$  of  $E$  is the largest size of an independent set contained in  $F$ . (Recall that  $E$  is assumed to be finite.) Note that the empty set is always an independent set contained in  $F$ . The rank of  $F$  is denoted by  $\rho_M(F)$ , and  $\rho_M$  is called the *rank function* of  $M$ .

A notation similar to one used for graphs will be adopted in the sequel concerning adding an element  $e$  to the set  $F$  (denoted by  $F + e$ ) or removing it from  $F$  (denoted by  $F - e$ ). Two easy properties of the rank function are the following

**Theorem 7.1.** *If  $M$  is a hereditary system of the set  $E$  then*

- (i)  $\rho_M(\emptyset) = 0$ , and
- (ii) for any subset  $F$  of  $E$  and any element  $e$ ,

$$\rho_M(F) \leq \rho_M(F + e) \leq \rho_M(F) + 1.$$

*Proof.* Item (i) is clear, so let us move to item (ii).

Since  $F + e$  contains those independent sets that are contained in  $F$ , we have  $\rho_M(F + e) \geq \rho_M(F)$ . On the other hand, possible independent subsets of  $F + e$  not contained in  $F$  may only consist of an independent subset of  $F$  and  $e$ , so  $\rho_M(F + e) \leq \rho_M(F) + 1$ .  $\square$

A hereditary system  $M$  may of course be specified by giving its independent sets, that is by giving  $\mathcal{I}_M$ . It can be specified as well by giving its bases, i.e.  $\mathcal{B}_M$ , independent sets will then be exactly all subsets of bases. On the other hand,  $M$  can be specified by giving its circuits, i.e.  $\mathcal{C}_M$ , independent sets are then the sets not containing circuits. Finally,  $M$  can be defined by giving the rank function  $\rho_M$ , since a set  $F$  is independent exactly when  $\rho_M(F) = \#(F)$ . (As before, we denote cardinality of a set  $F$  by  $\#(F)$ .) Thus an aspect may involve any of  $\mathcal{I}_M$ ,  $\mathcal{B}_M$ ,  $\mathcal{C}_M$  and  $\rho_M$ .

It might be mentioned that a hereditary system is a far too general concept to be of much use. This means that well chosen aspects are needed to restrict the concept to a more useful one (that is, a matroid). Let us have a look at certain proper aspects in connection with a matroid well familiar from the preceding chapters.

## 7.2 The Circuit Matroid of a Graph

The *circuit matroid*  $M(G)$  of a graph  $G = (V, E)$  is a hereditary system of the edge set  $E$  whose circuits are the circuits of  $G$ , considered as edge sets. (It is naturally assumed that  $G$  is not empty.) The bases of  $M(G)$  are the maximal independent edge sets, i.e. spanning forests of  $G$ , and the independent sets of  $M(G)$  are the subforests, both considered as edge sets. Let us denote  $G_F = (V, F)$  for a subset  $F$  of  $E$ . The number of vertices of  $G$  is denoted by  $n$ , as usual.

**Remark.** A hereditary system that is not directly a circuit matroid of any graph but has a structure identical to one is called a graphic matroid.

Let us then take a look at different aspects of the circuit matroid.

### Basis Exchange Property

Let us consider two bases (i.e. spanning forests)  $B_1$  and  $B_2$ . If  $e$  is an edge in  $B_1$ , its removal divides some component  $G'$  of the graph  $G$  into two disjoint subgraphs. Now certain edges of  $B_1$  will be the branches of a spanning tree  $T_1$  of  $G'$ , and similarly, certain edges in  $B_2$  will be the branches of a spanning tree  $T_2$  of  $G'$ . The removed edge  $e$  is either a branch of  $T_2$  or then a link of  $T_2^*$ . In the latter case  $e$  will be in the fundamental cut set determined by a branch  $f$  of  $T_2$  (cf. Theorem 2.7). Then  $T_1 - e + f$  is also a spanning tree of  $G'$  and we can replace  $e$  by  $f$  and get again a spanning forest of  $G$ , that is, a basis.

Hence we have

**Basis Exchange Property:** *If  $B_1$  and  $B_2$  are different bases and  $e \in B_1 - B_2$  then there is an element  $f \in B_2 - B_1$  such that  $B_1 - e + f$  is a basis.*

In general, a hereditary system with the basis exchange property will be a matroid. In other words, the basis exchange property is a proper aspect. Using basis exchange one can move from one basis to another. All bases are thus of the same size.

### Uniformity. Absorptivity

For a subset  $F$  of  $E$  let us denote by  $n_F$  the number of vertices in the subgraph  $\langle F \rangle$  of  $G$  induced by  $F$ , and by  $k_F$  the number of its components. Then there are  $n_F - k_F$  edges in a spanning forest of  $\langle F \rangle$ . Let us denote further by  $K_F$  the number of components of the subgraph  $G_F$  of  $G$ . Clearly then

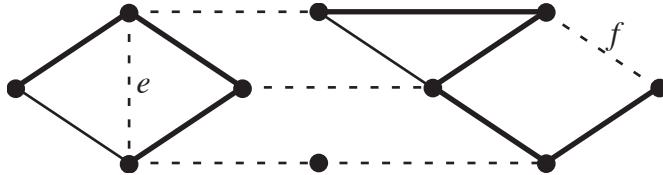
$$\rho_{M(G)}(F) = n_F - k_F = n - K_F,$$

and all such spanning forests are of the same size. Hence

**Uniformity:** *For a subset  $F$  of  $E$  all maximal independent subsets of  $F$  are of the same size. (Maximality of a set  $H$  means here that there are no independent sets  $J$  such that  $H \subset J \subseteq F$ .)*

In general, a hereditary system with the uniformity property will be a matroid, and uniformity is a proper aspect.

In the figure below continuous lines are the edges of  $F$ , with the thick ones being the branches of a spanning forest. Dashed lines indicate the remaining edges in  $E$ .



If  $e$  is an edge of  $G$  and  $\rho_{M(G)}(F + e) = \rho_{M(G)}(F)$  then  $e$  does not connect two components of  $G_F$ . Suppose  $f$  is another edge with the same property, that is,  $\rho_{M(G)}(F + f) = \rho_{M(G)}(F)$ . Clearly then

$$\rho_{M(G)}(F + e + f) = \rho_{M(G)}(F).$$

Thus we get

**Weak Absorptivity:** *If  $e, f \in E$  and  $F \subseteq E$  and*

$$\rho_M(F) = \rho_M(F + e) = \rho_M(F + f)$$

then also

$$\rho_M(F + e + f) = \rho_M(F).$$

In general a weakly absorptive hereditary system is a matroid, and thus weak absorptivity is another proper aspect.

By repeating the above argument sufficiently many times we see that if  $F$  and  $F'$  are sets of edges of  $G$ , and for each edge  $e$  in  $F'$  we have  $\rho_{M(G)}(F + e) = \rho_{M(G)}(F)$ , then

$$\rho_{M(G)}(F \cup F') = \rho_{M(G)}(F).$$

Hence also

**Strong Absorptivity:** If  $F, F' \subseteq E$  and  $\rho_M(F + e) = \rho_M(F)$  for each element  $e$  in  $F'$  then

$$\rho_M(F \cup F') = \rho_M(F).$$

We conclude that strong absorptivity is a proper aspect.

### Augmentation

Suppose  $I_1$  and  $I_2$  are independent sets of the circuit matroid  $M(G)$  (edge sets of subforests of  $G$ ) and  $\#(I_1) < \#(I_2)$ . The subgraph  $G_{I_1}$  then has  $n - \#(I_1)$  components, and the subgraph  $G_{I_2}$  has  $n - \#(I_2)$  components, so strictly less than  $G_{I_1}$ . Adding an edge does not reduce the number of components exactly in the case where the edge is added in some component. Thus, if adding any edge in  $I_2 - I_1$  to  $G_{I_1}$  preserves the number of components then it must be that the edge is added in some component of  $G_{I_1}$ , and  $G_{I_2}$  cannot have fewer components than  $G_{I_1}$ . But as noted, this is not the case if  $\#(I_1) < \#(I_2)$ , and so

**Augmentation:** If  $I_1$  and  $I_2$  are independent sets of the hereditary system  $M$  and  $\#(I_1) < \#(I_2)$  then there exists an element  $e \in I_2 - I_1$  such that  $I_1 + e$  is in  $\mathcal{I}_M$ .

In general, a hereditary system with the augmentation property is a matroid. Thus augmentation is a proper aspect, too.

### Elimination

The circuits of the circuit matroid  $M(G)$  are the edge sets of the circuits of  $G$ . The degree of a vertex in a circuit is two. If  $C_1$  and  $C_2$  are different circuits of  $M(G)$  then the degree of a vertex of the ring sum  $\langle C_1 \rangle \oplus \langle C_2 \rangle$  is also even, see Section 1.3. Hence  $\langle C_1 \rangle \oplus \langle C_2 \rangle$  must contain at least one circuit as a subgraph, since a ring sum does not have isolated vertices and a nonempty forest has at least one pending vertex (Theorem 2.3). Recalling the definition of ring sum in Section 1.3 it is noticed that such a circuit does not contain edges in the intersection  $C_1 \cap C_2$ , at least not with as high multiplicity as in  $C_1 \cup C_2$ . Thus

**Elimination Property:** If  $C_1$  and  $C_2$  are different circuits of the hereditary system  $M$  and  $e \in C_1 \cap C_2$  then there is a circuit  $C \in \mathcal{C}_M$  such that  $C \subseteq C_1 \cup C_2 - e$ .

Again, elimination property is a proper aspect, and a hereditary system with the elimination property is a matroid.

### Induced Circuits

If  $I$  is an independent set of the circuit matroid  $M(G)$  (edge set of a subforest) then adding one edge either closes exactly one circuit in a component of  $G_I$  (Theorem 2.3), or then it connects two components of  $G_I$  and does not create a circuit. We have then

**Property of Induced Circuits:** *If  $I$  is an independent set of a hereditary system  $M$  and  $e \in E$  then  $I + e$  contains at most one circuit.*

The property of induced circuits is a proper aspect, and a hereditary system having this property will be a matroid.

## 7.3 Other Basic Matroids

### Vectorial Matroid

Let  $E$  be a finite set of vectors of a vector space (say  $\mathbb{R}^n$ ) and the independent sets of a hereditary system  $M$  of  $E$  be exactly all linearly independent subsets of  $E$  (including the empty set).  $M$  is then a so-called *vectorial matroid*. Here  $E$  is usually allowed to be a multiset, i.e. its elements have multiplicities—cf. parallel edges of graphs. It is then agreed, too, that a subset of  $E$  is linearly dependent when one its elements has a multiplicity higher than one. A hereditary system that is not directly vectorial but is structurally identical to a vectorial matroid  $M'$  is called a *linear matroid*, and the matroid  $M'$  is called its *representation*.

A circuit of a vectorial matroid is a linearly dependent set  $C$  of vectors such that removing any of its elements leaves a linearly independent set—keeping in mind possible multiple elements. An aspect typical to vectorial matroids is the elimination property. If  $C_1 = \{\mathbf{r}, \mathbf{r}_1, \dots, \mathbf{r}_k\}$  and  $C_2 = \{\mathbf{r}, \mathbf{r}'_1, \dots, \mathbf{r}'_l\}$  are different circuits sharing (at least) the vector  $\mathbf{r}$  then  $\mathbf{r}$  can be represented as linear combinations of other vectors in both  $C_1$  and  $C_2$ , and in such a way that all coefficients in the combinations are nonzero. We get thus an equality

$$\sum_{i=1}^k c_i \mathbf{r}_i - \sum_{j=1}^l c'_j \mathbf{r}'_j = \mathbf{0}.$$

Combining (possible) repetitive vectors on the left hand side, and noticing that this does not make it empty, we see that  $C_1 \cup C_2 - \mathbf{r}$  contains a circuit. (Note especially the case where either  $C_1 = \{\mathbf{r}, \mathbf{r}\}$  or  $C_2 = \{\mathbf{r}, \mathbf{r}\}$ .)

In the special case where  $E$  consists of columns (or rows) of a matrix  $\mathbf{A}$ , a vectorial matroid of  $E$  is called a *matrix matroid* and denoted by  $M(\mathbf{A})$ . For example, the circuit matroid  $M(G)$  of a graph  $G$  is a linear matroid whose representation is obtained using the rows of the circuit matrix of  $G$  in the binary field  $GF(2)$  (see Section 4.5).<sup>3</sup> Of course, if desired, any vectorial matroid of  $E$  may be considered as a matrix matroid simply by taking the vectors of  $E$  as columns (or rows) of a matrix.<sup>4</sup>

---

<sup>3</sup>Hereditary systems with a representation in the binary field  $GF(2)$  are called *binary matroids*. The circuit matroid of a graph is thus always binary.

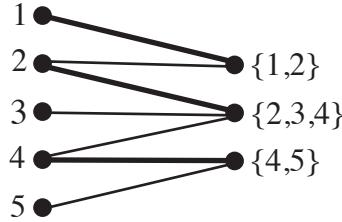
<sup>4</sup>This actually is the origin of the name "matroid". A matroid is a generalization of a linear matroid and a linear matroid may be thought of as a matrix. Indeed, not all matroids are linear. The name "matroid" was strongly opposed at one time. Even today there are people who prefer to use names like "geometry" or "combinatorial geometry".

### Transversal Matroid

Let  $\mathcal{A} = \{A_1, \dots, A_k\}$  be a family of nonempty finite sets. The *transversal matroid*  $M(\mathcal{A})$  is a hereditary system of the set  $E = A_1 \cup \dots \cup A_k$  whose independent sets are exactly all subsets of  $E$  containing at most one element of each of the sets  $A_i$  (including the empty set). Here it is customary to allow the family  $\mathcal{A}$  to be a multiset, that is, a set  $A_i$  may appear several times as its element, thus allowing more than one element of  $A_i$  in an independent set.

A natural aspect of transversal matroids is augmentation, and it is connected with augmentings of matchings of bipartite graphs! (See Section 5.8.) Let us define a bipartite graph  $G = (V, E')$  as follows: The vertex set is  $V = E \cup \mathcal{A}$ , and the vertices  $e$  and  $A_j$  are connected by an edge exactly when  $e \in A_j$ . (Note how the vertex set  $V$  is naturally divided into the two parts of the cut,  $E$  and  $\mathcal{A}$ .) An independent set of  $M(\mathcal{A})$  is then a set of matched vertices of  $G$  in  $E$ , and vice versa.

**Example.** In the figure below is the bipartite graph corresponding to the transversal matroid of the family  $\{\{1, 2\}, \{2, 3, 4\}, \{4, 5\}\}$ , and its independent set  $\{1, 2, 4\}$  (thick line).



Very much in the same way as in the proof of Theorem 5.3 one may show that if  $I_1$  and  $I_2$  are independent sets (vertex sets of the matchings  $S_1$  and  $S_2$ ) and  $\#(I_1) < \#(I_2)$  then there is an augmenting path of the matching  $S_1$  such that the new matched vertex is in  $I_2$ . Thus  $M(\mathcal{A})$  indeed has the augmentation property.

**Remark.** For matchings of bipartite graphs the situation is completely general. That is, matchings of bipartite graphs can always be thought of as independent sets of transversal matroids. In fact this remains true for matchings of general graphs, too, leading to the so-called matching matroids, see e.g. SWAMY & THULASIRAMAN.

If the sets of the family  $\mathcal{A}$  are disjoint—i.e. they form a partition of  $E$ —then the transversal matroid is also called *partition matroid*. For a partition matroid augmentation is obvious.

### Uniform Matroid

For all finite sets  $E$  one can define the so-called *uniform matroids*. The uniform matroid of  $E$  of rank  $k$ , denoted  $U_k(E)$ , is a hereditary system whose independent sets are exactly all subsets of  $E$  containing at most  $k$  elements. The bases of  $U_k(E)$  are those subsets containing exactly  $k$  elements, and the circuits are the subsets containing exactly  $k + 1$  elements. In particular, all subsets of  $E$  form a uniform matroid of  $E$  of rank  $\#(E)$ , this is often called the *free matroid* of  $E$ . Quite obviously  $U_k(E)$  has the basis exchange property and the augmentation property.

Uniform matroids are not very interesting as such. They can be used as "building blocks" of much more complicated matroids, however. It may also be noted that uniform matroids are transversal matroids (can you see why?).

## 7.4 Greedy Algorithm

Many problems of combinatorial optimization<sup>5</sup> may be thought of as finding a heaviest or a lightest independent set of a hereditary system  $M$  of  $E$ , when each element of  $E$  is given a weight. The weighting function is  $\alpha : E \rightarrow \mathbb{R}$  and the weight of a set  $F \subseteq E$  is

$$\sum_{e \in F} \alpha(e).$$

The two optimization modes are interchanged when the signs of the weights are reversed.

One may also find the heaviest or the lightest bases. Again reversing the signs of the weights interchanges maximization and minimization. If all bases are of the same size—as will be the case for matroids—they can be restricted to the case where there weights are positive. Indeed, if  $A$  is the smallest weight of an element of  $E$  then changing the weight function to

$$\beta : \beta(e) = 1 + \alpha(e) - A$$

one gets an equivalent optimization problem with positive weights. On the other hand, maximization and minimization are interchanged when the weighting function is changed to

$$\beta : \beta(e) = 1 + B - \alpha(e)$$

where  $B$  is the largest weight of an element of  $E$ .

**Example.** (A bit generalized) Kruskal's Algorithm (see Section 5.6) finds a lightest spanning forest of an edge-weighted graph  $G$ , i.e. a lightest basis of the circuit matroid of  $G$ . As was seen, this can be done quite fast—and even faster if the edges are given in the order of increasing weight when one can always consider the "best" remaining edge to be included in the forest. Kruskal's Algorithm No. 1 is an example of a so-called greedy algorithm that always proceeds in the "best" available direction. Such a greedy algorithm is fast, indeed, it only needs to find this "best" element to be added in the set already constructed.

It might be mentioned that Kruskal's Algorithm No. 3 is also a greedy algorithm, it finds a heaviest cospanning forest in the dual matroid of the circuit matroid, the so-called bond matroid of  $G$  (see Section 7.6).

Even though greedy algorithms produce the correct result for circuit matroids they do not always do so.

**Example.** Finding a lightest Hamiltonian circuit of an edge-weighted graph  $G$  may also be thought of as finding the lightest basis of a hereditary system—assuming of course that there are Hamiltonian circuits. The set  $E$  is again taken to be the edge set of  $G$  but now the bases are the Hamiltonian circuits of  $G$  (considered as edge sets). A lightest basis is then a lightest Hamiltonian circuit. As was noticed in Section 5.7, finding a lightest Hamiltonian circuit is a well-known  $\mathcal{NP}$ -complete problem and no greedy algorithm can thus always produce a (correct) result—at least if  $\mathcal{P} \neq \mathcal{NP}$ . The hereditary system thus obtained is in general not a matroid, however (e.g. it does not generally have the basis exchange property).

It would thus appear that—at least for matroids—greedy algorithms are favorable methods for finding heaviest/lightest bases (or independent sets). Indeed, matroids are precisely those hereditary systems for which this holds true. To be able to proceed further we define the *greedy algorithm* formally. We consider first maximization of independent sets, minimization is given in brackets. The input is a hereditary system  $M$  of the set  $E$ , and a weighting function  $\alpha$ .

---

<sup>5</sup>These problems are dealt with more extensively in the course Optimization Theory 2.

**Greedy Algorithm for Independent Sets:**

1. Sort the elements  $e_1, \dots, e_m$  of  $E$  according to decreasing [increasing] weight:  $e_{(1)}, \dots, e_{(m)}$ .
2. Set  $F \leftarrow \emptyset$  and  $k \leftarrow 1$ .
3. If  $\alpha(e_{(k)}) \leq 0$  [ $\alpha(e_{(k)}) \geq 0$ ], return  $F$  and quit.
4. If  $\alpha(e_{(k)}) > 0$  [ $\alpha(e_{(k)}) < 0$ ] and  $F \cup \{e_{(k)}\}$  is independent, set  $F \leftarrow F \cup \{e_{(k)}\}$ .
5. If  $k = m$ , return  $F$  and quit. Else set  $k \leftarrow k + 1$  and go to #3.

For bases the algorithm is even simpler:

**Greedy Algorithm for Bases:**

1. Sort the elements  $e_1, \dots, e_m$  of  $E$  according to decreasing [increasing] weight:  $e_{(1)}, \dots, e_{(m)}$ .
2. Set  $F \leftarrow \emptyset$  and  $k \leftarrow 1$ .
3. If  $F \cup \{e_{(k)}\}$  is independent, set  $F \leftarrow F \cup \{e_{(k)}\}$ .
4. If  $k = m$ , return  $F$  and quit. Else set  $k \leftarrow k + 1$  and go to #3.

The main result that links working of greedy algorithms and matroids is

**Theorem 7.2. (Matroid Greediness Theorem)** *The greedy algorithm produces a correct heaviest independent set of a hereditary system for all weight functions if and only if the system is a matroid. (This is the so-called greediness property.) The corresponding result holds true for bases, and also for finding lightest independent sets and bases. Furthermore, in both cases it suffices to consider positive weights.*

*Proof.* The first sentence of the theorem is proved as part of the proof of Theorem 7.3 in the next section.

As noted above, greediness is equivalent for maximization and minimization, for both independent sets and bases. It was also noted that finding a heaviest basis may be restricted to the case of positive weights. Since for positive weights a heaviest independent set is automatically a basis, greediness for bases follows from greediness for independent sets.

On the other hand, if greediness holds for bases, it holds for independent sets as well. Maximization of independent sets using the weight function  $\alpha$  then corresponds to maximization of bases for the positive weight function

$$\beta : \beta(e) = 1 + \max(0, \alpha(e)),$$

the greedy algorithms behave exactly similarly, item #3 is not activated for independent sets. Elements of weight 1 should be removed from the output.  $\square$

**Remark.** *Greediness is thus also a proper aspect for matroids. For hereditary families of sets it is equivalent to usefulness of the greedy algorithm. Certain other similar but more general families of sets have their own "greediness theorems". Examples are the so-called greedoids and matroid embeddings.*

## 7.5 The General Matroid

Any one of the several aspects above makes a hereditary system a matroid. After proving that they are all equivalent, we may define a *matroid* as a hereditary system that has (any) one of these aspects.

Before that we add one aspect to the list, which is a bit more difficult to prove directly for circuits matroids of graphs:

**Submodularity:** *If  $M$  is a hereditary system of the set  $E$  and  $F, F' \subseteq E$  then*

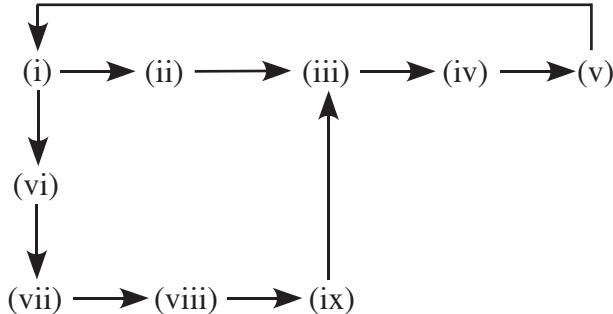
$$\rho_M(F \cap F') + \rho_M(F \cup F') \leq \rho_M(F) + \rho_M(F').$$

Let us then prove the equivalences, including submodularity.

**Theorem 7.3.** *If a hereditary system has (any) one of the nine aspects below then it has them all (and is a matroid).*

- |                              |                                     |
|------------------------------|-------------------------------------|
| (i) Uniformity               | (vi) Submodularity                  |
| (ii) Basis exchange property | (vii) Elimination property          |
| (iii) Augmentation property  | (viii) Property of induced circuits |
| (iv) Weak absorptivity       | (ix) Greediness                     |
| (v) Strong absorptivity      |                                     |

*Proof.* The implications are proved following the strongly connected digraph below:



All nine aspects are then connected by implication chains in both directions, and are thus logically equivalent. Let us consider a general hereditary system  $M$  of the set  $E$ .

(i) $\Rightarrow$ (ii): As a consequence of uniformity, all bases of  $M$  are of the same size. If  $B_1, B_2 \in \mathcal{B}_M$  and  $e \in B_1 - B_2$ , we may apply uniformity to the set  $F = (B_1 - e) \cup B_2$ . All maximal independent sets included in  $F$  are then of the same size as  $B_2$  (and  $B_1$ ). Now  $B_1 - e$  is not one of these maximal sets having too few elements. On the other hand, by adding one element  $f$  to  $B_1 - e$  we get such an independent set  $H$ . The element  $f$  must then be in the set difference  $B_2 - B_1$ , so  $H = B_1 - e + f$ . Moreover,  $H$  has as many elements as  $B_1$ , and so it is a basis.

(ii) $\Rightarrow$ (iii): If  $I_1, I_2 \in \mathcal{I}_M$  and  $\#(I_1) < \#(I_2)$ , we choose bases  $B_1$  and  $B_2$  such that  $I_1 \subseteq B_1$  and  $I_2 \subseteq B_2$ . Applying basis exchange (repeatedly) we replace those elements of  $B_1 - I_1$  that are not in  $B_2$  by elements of  $B_2$ . After this operation we may assume that  $B_1 - I_1 \subseteq B_2$ . As a consequence of the basis exchange property all bases are of the same size. Thus

$$\#(B_1 - I_1) = \#(B_1) - \#(I_1) > \#(B_2) - \#(I_2) = \#(B_2 - I_2),$$

and  $B_1 - I_1$  cannot be included in  $B_2 - I_2$ . Therefore there is an element  $e$  of  $B_1 - I_1$  in  $I_2$  and  $I_1 + e$  is an independent set.

(iii) $\Rightarrow$ (iv): Let us consider a situation where

$$\rho_M(F) = \rho_M(F + e) = \rho_M(F + f).$$

If now  $\rho_M(F + e + f) > \rho_M(F)$ , we take a maximal independent subset  $I_1$  of  $F$  and a maximal independent subset  $I_2$  of  $F + e + f$ . Then  $\#(I_2) > \#(I_1)$  and by the augmentation property  $I_1$  can be augmented by an element of  $I_2$ . This element cannot be in  $F$  (why not?), so it must be either  $e$  or  $f$ . But then  $\rho_M(F) < \rho_M(F + e)$  or  $\rho_M(F) < \rho_M(F + f)$  ( $\checkmark$ ).

(iv) $\Rightarrow$ (v): Let us assume weak absorptivity and consider subsets  $F$  and  $F'$  of  $E$  such that  $\rho_M(F + e) = \rho_M(F)$  for each element  $e$  of  $F'$ . We use induction on  $k = \#(F' - F)$  and show that  $\rho_M(F) = \rho_M(F \cup F')$  (strong absorptivity).

Induction Basis: Now  $k = 0$  or  $k = 1$  and the matter is clear.

Induction Hypothesis: The claimed result holds true when  $k \leq \ell$ . ( $\ell \geq 1$ )

Induction Statement: The claimed result holds true when  $k = \ell + 1$ .

Induction Statement Proof: Choose distinct elements  $e, f \in F' - F$  and denote  $F'' = F' - e - f$ . The Induction Hypothesis implies that

$$\rho_M(F) = \rho_M(F \cup F'') = \rho_M(F \cup F'' + e) = \rho_M(F \cup F'' + f).$$

Applying weak absorptivity to this it is seen that

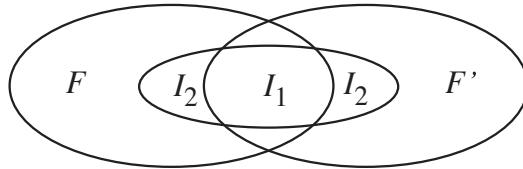
$$\rho_M(F) = \rho_M(F \cup F'' + e + f) = \rho_M(F \cup F').$$

(v) $\Rightarrow$ (i): If  $I$  is a maximal independent subset of  $F$  then  $\rho_M(I + e) = \rho_M(I)$  for elements  $e$  in the set difference  $F - I$  (if any). Strong absorptivity implies then that  $\rho_M(F) = \rho_M(I) = \#(I)$ , i.e. all these independent sets are of the same size and uniformity holds true.

(i) $\Rightarrow$ (vi): Let us consider sets  $F, F' \subseteq E$  and denote by  $I_1$  a maximal independent subset of the intersection  $F \cap F'$  and by  $I_2$  a maximal independent subset of the union  $F \cup F'$ . Uniformity implies augmentation, so we may assume that  $I_2$  is obtained from  $I_1$  by adding elements, that is  $I_1 \subseteq I_2$ . Now  $I_2 \cap F$  is an independent subset of  $F$  and  $I_2 \cap F'$  is an independent subset of  $F'$ , and both of them include  $I_1$ . So

$$\begin{aligned} \rho_M(F \cap F') + \rho_M(F \cup F') &= \#(I_1) + \#(I_2) \\ &\stackrel{*}{=} \#(I_2 \cap F) + \#(I_2 \cap F') \leq \rho_M(F) + \rho_M(F'). \end{aligned}$$

The equality marked by an asterisk is a set-theoretical one, see the figure below.



(vi) $\Rightarrow$ (vii): Let us consider distinct circuits  $C_1, C_2 \in \mathcal{C}_M$  and an element  $e \in C_1 \cap C_2$ . Then  $\rho_M(C_1) = \#(C_1) - 1$  and  $\rho_M(C_2) = \#(C_2) - 1$ , and  $\rho_M(C_1 \cap C_2) = \#(C_1 \cap C_2)$ . (Remember that every proper subset of a circuit is independent.) If now  $C_1 \cup C_2 - e$  does not contain a circuit, it is independent and  $\rho_M(C_1 \cup C_2 - e) = \#(C_1 \cup C_2) - 1$ , whence  $\rho_M(C_1 \cup C_2) \geq \#(C_1 \cup C_2) - 1$ . Submodularity however implies that

$$\rho_M(C_1 \cap C_2) + \rho_M(C_1 \cup C_2) \leq \rho_M(C_1) + \rho_M(C_2),$$

and further that (check!)

$$\#(C_1 \cap C_2) + \#(C_1 \cup C_2) \leq \#(C_1) + \#(C_2) - 1.$$

This is a set-theoretical impossibility, and thus  $C_1 \cup C_2 - e$  does contain a circuit.

(vii) $\Rightarrow$ (viii): If  $I$  is an independent set and  $I + e$  contains two distinct circuits  $C_1$  and  $C_2$  then obviously both  $C_1$  and  $C_2$  contain the element  $e$ . The elimination property implies that  $C_1 \cup C_2 - e$  contains a circuit. Since  $C_1 \cup C_2 - e$  is however contained in  $I$ , it is independent ( $\checkmark$ ). So  $I + e$  contains at most one circuit.

(viii) $\Rightarrow$ (ix): Let us denote by  $I$  the output of the greedy algorithm for the weighting function  $\alpha$ . (The problem is finding a heaviest independent set.) If  $I$  is a heaviest independent set, then the matter is clear. Otherwise we take a heaviest independent set having the largest intersection with  $I$ . Let us denote this heaviest independent set by  $I'$ .  $I$  cannot be a subset of  $I'$ , because the greedy algorithm would then find an even heavier independent set. Let us further denote by  $e$  the first element of the set difference  $I - I'$  that the greedy algorithm chooses.  $I' + e$  is a dependent set and contains thus exactly one circuit  $C$  (remember the property of induced circuits). This circuit of course is not included in  $I$ , so there is an element  $f \in C - I$ . Since  $I' + e$  contains only one circuit,  $I' + e - f$  is an independent set.  $I'$  is maximal, so that  $\alpha(f) \geq \alpha(e)$ . On the other hand,  $f$  and those elements of  $I$  that the greedy algorithm chose before choosing  $e$  are all in  $I'$ , whence adding  $f$  to the elements does not create a circuit. This means that  $f$  was available for the greedy algorithm when it chose  $e$ , and so  $\alpha(f) \leq \alpha(e)$ . We conclude that  $\alpha(f) = \alpha(e)$  and the sets  $I' + e - f$  and  $I'$  have equal weight. This however is contrary to the choice of  $I'$  because  $\#((I' + e - f) \cap I) > \#(I' \cap I)$ . (The reader may notice a similarity to the proof of Theorem 5.2. Indeed, this gives another proof for Kruskal's Algorithm No. 1.)

(ix) $\Rightarrow$ (iii): Let us consider independent sets  $I_1$  and  $I_2$  such that  $\#(I_1) < \#(I_2)$ . For brevity we denote  $k = \#(I_1)$ . Consider then the weighting function

$$\alpha : \alpha(e) = \begin{cases} k+2, & \text{if } e \in I_1 \\ k+1, & \text{if } e \in I_2 - I_1 \\ 0 & \text{otherwise.} \end{cases}$$

The weight of  $I_2$  is then

$$\sum_{e \in I_2} \alpha(e) \geq (k+1)^2 > k(k+2) = \sum_{e \in I_1} \alpha(e).$$

It is thus larger than the weight of  $I_1$ , so  $I_1$  is not a heaviest independent set. On the other hand, when finding a heaviest independent set the greedy algorithm will choose all elements of  $I_1$  before it ever chooses an element of  $I_2 - I_1$ . Since it is now assumed to produce a heaviest independent set, it must choose at least one element  $e$  of  $I_2 - I_1$  and  $I_1 + e$  is thus an independent set. This shows that the augmentation property holds true.  $\square$

The most popular aspect defining a matroid is probably the augmentation property.

## 7.6 Operations on Matroids

In the preceding chapters, in connection with fundamental cut sets and fundamental circuits, mutual duality was mentioned. Duality is a property that is very natural for hereditary systems and matroids.

The *dual (system)*  $M^*$  of a hereditary system  $M$  of the set  $E$  is a hereditary system of  $E$  whose bases are the complements of the bases of  $M$  (against  $E$ ). Often the bases of  $M^*$  are called *cobases* of  $M$ , circuits of  $M^*$  are called *cocircuits* of  $M$ , and so on. It is easily checked that  $M^*$  really is a hereditary system of  $E$ : If  $\overline{B}_1$  and  $\overline{B}_2$  are distinct bases of  $M^*$  then  $B_1$  and  $B_2$  are distinct bases of  $M$ . Thus, if  $\overline{B}_1 \subseteq \overline{B}_2$  then  $B_2 \subseteq B_1$  ( $\checkmark$ ). Note also that  $(M^*)^* = M$ .

**Theorem 7.4. (Whitney's Theorem)** *The dual  $M^*$  of a matroid  $M$  is a matroid, the so-called dual matroid, and*

$$\rho_{M^*}(F) = \#(F) - \rho_M(E) + \rho_M(\overline{F}).$$

(Note that  $\rho_M(E)$  is the size of a basis of  $M$ .)

*Proof.* Let us show that  $M^*$  has the basis exchange property, which makes it a matroid according to Theorem 7.3. If  $\overline{B}_1$  and  $\overline{B}_2$  are distinct bases of  $M^*$  and  $e \in \overline{B}_1 - \overline{B}_2$  then  $B_1$  and  $B_2$  are distinct bases of  $M$  and  $e \in B_2 - B_1$ . Since  $B_1$  is a basis of  $M$ ,  $B_1 + e$  contains exactly one circuit  $C$  of  $M$  (the property of induced circuits) and this circuit must have an element  $f \in B_2 - B_1$ . Then however  $B_1 + e - f$  does not contain a circuit of  $M$ , i.e. it is an independent set of  $M$ , and has the same size as  $B_1$ . All bases have the same size, so  $B_1 + e - f$  is a basis of  $M$  and its complement  $\overline{B}_1 - e + f$  is a basis of  $M^*$ .

To compute the rank  $\rho_{M^*}(F)$  we take a maximal independent set  $H$  of  $M^*$  included in  $F$ . Then

$$\rho_{M^*}(F) = \rho_{M^*}(H) = \#(H).$$

Then  $\overline{H}$  is a minimal set containing the set  $\overline{F}$  and a basis of  $M$ . (This is simply the same statement in other words. Note that  $H$  is included in some basis of  $M^*$ .) But such a set is obtained starting from  $\overline{F}$ , taking a maximal independent set of  $M$  contained in  $\overline{F}$ —which has  $\rho_M(\overline{F})$  elements—and extending it to a basis—which has  $\rho_M(E)$  elements. So

$$\#(\overline{H}) - \#(\overline{F}) = \rho_M(E) - \rho_M(\overline{F}).$$

Set theory tells us that

$$\#(\overline{H}) + \#(H) = \#(E) = \#(\overline{F}) + \#(F).$$

Combining these we get the claimed formula for  $\rho_{M^*}(F)$  (check!). □

Dualism gives a connection between bases of a matroid  $M$  and circuits of its dual matroid  $M^*$  (i.e. cocircuits of  $M$ ):

**Theorem 7.5.** (i) *Circuits of the dual matroid of a matroid  $M$  are the minimal sets that intersect every basis of  $M$ .*

(ii) *Bases of a matroid  $M$  are the minimal sets that intersect every circuit of the dual matroid  $M^*$ .*

*Proof.* (i) The circuits of  $M^*$  are the minimal sets that are not contained in any complement of a basis of  $M$ . Thus they must intersect every basis of  $M$ .

(ii) Bases of  $M^*$  are the maximal sets that do not contain any circuit of  $M^*$ . The same in other words: Bases of  $M$  are the minimal sets that intersect every circuit of  $M^*$ . □

**Example.** *Bases of the circuit matroid  $M(G)$  of a connected graph  $G$  are the spanning trees. Bases of the dual matroid  $M^*(G)$  are the complements of these, i.e. the cospanning trees. By the theorem, circuits of the dual matroid are the cut sets of  $G$ . (Cf. Theorems 2.4 and 2.5.) Because according to Whitney's Theorem  $M^*(G)$  is a matroid, it has the greediness property, that is, the*

greedy algorithm finds a heaviest/lightest basis. Working of Kruskal's Algorithm No. 3 is based on this. The algorithm finds the heaviest cospanning tree.

Analogous concepts can naturally be defined for a general, possibly disconnected, graph  $G$ . Bases of  $M^*(G)$  are then the cospanning forests of  $G$ . The dual matroid  $M^*(G)$  is called the bond matroid or the cut matroid or the cocircuit matroid of  $G$ . So, when is the bond matroid  $M^*(G)$  graphic, i.e. the circuit matroid of a graph? The so-called Whitney Planarity Theorem tells us that this happens exactly when  $G$  is a planar graph! (See e.g. WEST.)

If  $M_i$  is a hereditary system of the set  $E_i$  for  $i = 1, \dots, k$  then the direct sum  $M = M_1 \oplus \dots \oplus M_k$  of the systems  $M_1, \dots, M_k$  is the hereditary system of the set  $E = E_1 \cup \dots \cup E_k$  whose independent sets are exactly all sets  $I_1 \cup \dots \cup I_k$  where  $I_i \in \mathcal{I}_{M_i}$  ( $i = 1, \dots, k$ ). In particular, if  $E_1 = \dots = E_k = E$  then the direct sum  $M$  is called the union of the systems  $M_1, \dots, M_k$ , denoted by  $M = M_1 \cup \dots \cup M_k$ . Note that each hereditary system  $M_i$  could also be thought of as a hereditary system of the set  $E$  simply by adding elements of  $E - E_i$  as circuits (loops, that is).

It is not exactly difficult to see that if  $M_1, \dots, M_k$  are matroids and the sets  $E_1, \dots, E_k$  are pairwise disjoint then  $M = M_1 \oplus \dots \oplus M_k$  is a matroid, say, by demonstrating the augmentation property (try it!). But actually a more general result holds true:

**Theorem 7.6. (Matroid Union Theorem)<sup>6</sup>** If  $M_1, \dots, M_k$  are matroids of the set  $E$  then the union  $M = M_1 \cup \dots \cup M_k$  is also a matroid of  $E$  and

$$\rho_M : \rho_M(F) = \min_{F' \subseteq F} \left( \#(F - F') + \sum_{i=1}^k \rho_{M_i}(F') \right).$$

*Proof.* The proof is rather long and difficult, and is not given here (see e.g. WEST or OXLEY). It might be mentioned, though, that the rank formula is not valid for hereditary systems in general.  $\square$

The theorem has many fundamental corollaries, e.g.

**Corollary. (Matroid Covering Theorem)<sup>7</sup>** If  $M$  is a loopless matroid of the set  $E$  then the smallest number of independent sets whose union equals  $E$  is

$$\max_{F \subseteq E} \left\lceil \frac{\#(F)}{\rho_M(F)} \right\rceil.$$

*Proof.* Note first that since  $M$  is loopless, each element of  $E$  is in itself an independent set. The set  $E$  thus can be covered as stated. Take now  $k$  copies of  $M$  as the matroids  $M_1, \dots, M_k$  in the union theorem. Then  $E$  is a union of  $k$  independent sets of  $M$  exactly when it is an independent set of the union matroid  $M' = M_1 \cup \dots \cup M_k$ . The covering property we are interested in can then be expressed in the form  $\rho_{M'}(E) = \#(E)$  or, by the union theorem,

$$\#(E) = \min_{F \subseteq E} \left( \#(E - F) + \sum_{i=1}^k \rho_{M_i}(F) \right)$$

i.e.

$$\min_{F \subseteq E} (k\rho_M(F) - \#(F)) = 0.$$

Since the difference to be minimized is  $= 0$  when  $F$  is the empty set,  $k$  will be the smallest number such that  $k \geq \#(F)/\rho_M(F)$  for all nonempty subsets  $F \subseteq E$ .  $\square$

---

<sup>6</sup>Also known by the names Edmonds–Fulkerson Theorem and Matroid Sum Theorem.

<sup>7</sup>Also known as Edmonds' Covering Theorem.

**Example.** For the circuit matroid  $M(G)$  of a loopless graph  $G$  independent sets are the subforests of  $G$ , and we are interested in the minimum number of subforests needed to contain all edges of  $G$ . Let us denote this number by  $A(G)$ , it is called the arboricity of  $G$ .

To analyze the maximization in the covering theorem we divide the subgraph  $\langle F \rangle$  induced by the edges in  $F$  into its components. Numbers of vertices and edges of these components are denoted by  $n_1, \dots, n_{k_F}$  and  $m_1, \dots, m_{k_F}$ , respectively. We use an indexing such that

$$\frac{m_{k_F}}{n_{k_F} - 1} \geq \frac{m_{k_F-1}}{n_{k_F-1} - 1} \geq \dots \geq \frac{m_1}{n_1 - 1}.$$

Now, in general if  $\frac{x_2}{y_2} \geq \frac{x_1}{y_1}$  then  $\frac{x_2}{y_2} \geq \frac{x_1 + x_2}{y_1 + y_2}$ . Thus

$$\frac{m_2}{n_2 - 1} \geq \frac{m_1 + m_2}{n_1 + n_2 - 2},$$

and continuing inductively, also

$$\frac{m_i}{n_i - 1} \geq \frac{m_1 + \dots + m_i}{n_1 + \dots + n_i - i} \quad (i = 1, \dots, k_F).$$

In particular then

$$\frac{m_{k_F}}{n_{k_F} - 1} \geq \frac{m_1 + \dots + m_{k_F}}{n_1 + \dots + n_{k_F} - k_F} = \frac{\#(F)}{\rho_{M(G)}(F)}.$$

Maximization can thus be restricted to edge-sets  $F$  such that  $\langle F \rangle$  is connected and  $\rho_{M(G)}(F) = n_F - 1$  where  $n_F$  is the number of vertices of  $F$ . (It might be further restricted to edge-sets  $F$  such that  $\langle F \rangle$  also equals the subgraph induced by its vertices, since connecting two vertices by an edge increases the numerator of the fraction to be maximized, the denominator remaining the same.) Thus we get the celebrated Nash-Williams Formula for arboricity:

$$A(G) = \max_{F \subseteq E} \left\lceil \frac{\#(F)}{n_F - 1} \right\rceil.$$

It might be noted that since for a simple planar graph  $\#(F) \leq 3n_F - 6$  (Linear Bound applied to  $\langle F \rangle$ ),  $A(G)$  is then at most 3.

The *restriction* of a hereditary system  $M$  of the set  $E$  into the set  $F \subseteq E$  is a hereditary system  $M|F$  whose independent sets are exactly those subsets of  $F$  that are independent sets of  $M$ . The *contraction* of  $M$  into the set  $F$  is the hereditary system  $(M^*|F)^*$ , often denoted by  $M.F$ . Clearly the augmentation property of  $M$  is directly transferred to  $M|F$ , so (cf. Whitney's Theorem)

**Theorem 7.7.** If  $M$  is a matroid of the set  $E$  and  $F \subseteq E$  then  $M|F$  and  $M.F$  are both matroids, too.

The *minors* of a matroid  $M$  are all those matroids that can be obtained from  $M$  by consecutive restrictions and contractions.

# References

1. ANDRÁSFAI, B.: *Introductory Graph Theory*. The Institute of Physics (1978)
2. ANDRÁSFAI, B.: *Graph Theory: Flows, Matrices*. The Institute of Physics (1991)
3. BANG-JENSEN, J. & GUTIN, G.: *Digraphs: Theory, Algorithms and Applications*. Springer–Verlag (2002)
4. BOLLOBÁS, B.: *Modern Graph Theory*. Springer–Verlag (2002)
5. CHRISTOFIDES, N.: *Graph Theory. An Algorithmic Approach*. Academic Press (1975)
6. DIESTEL, R.: *Graph Theory*. Springer–Verlag (2005)
7. DOLAN, A. & ALDOUS, J.: *Networks and Algorithms. An Introductory Approach*. Wiley (1999)
8. GIBBONS, A.: *Algorithmic Graph Theory*. Cambridge University Press (1987)
9. GIBBONS, A. & RYTTER, W.: *Efficient Parallel Algorithms*. Cambridge University Press (1990)
10. GONDTRAN, M. & MINOUX, M.: *Graphs and Algorithms*. Wiley (1986)
11. GRIMALDI, R.P.: *Discrete and Combinatorial Mathematics*. Addison–Wesley (2003)
12. GROSS, J. & YELLEN, J.: *Graph Theory and Its Applications*. CRC Press (2006)
13. GROSS, J. & YELLEN, J.: *Handbook of Graph Theory*. CRC Press (2003)
14. HOPCROFT, J.E. & ULLMAN, J.D.: *Introduction to Automata Theory, Languages, and Computation*. Addison–Wesley (1979)
15. JUNGNICKEL, D.: *Graphs, Networks and Algorithms*. Springer–Verlag (2004)
16. MCCLIECE, R.J. & ASH, R.B. & ASH, C.: *Introduction to Discrete Mathematics*. McGraw–Hill (1990)
17. MCHUGH, J.A.: *Algorithmic Graph Theory*. Prentice–Hall (1990)
18. MEHLHORN, K.: *Graph Algorithms and NP-Completeness*. Springer–Verlag (1984)
19. NOVAK, L. & GIBBONS, A.: *Hybrid Graph Theory and Network Analysis*. Cambridge University Press (1999)
20. OXLEY, J.G.: *Matroid Theory*. Oxford University Press (2006)

21. READ, R.C. & WILSON, R.J.: *An Atlas of Graphs*. Oxford University Press (2004)
22. SKIENA, S.S.: *The Algorithm Design Manual*. Springer–Verlag (1998)
23. SWAMY, M.N.S. & THULASIRAMAN, K.: *Graphs, Networks, and Algorithms*. Wiley (1981)
24. SWAMY, M.N.S. & THULASIRAMAN, K.: *Graphs: Theory and Algorithms*. Wiley (1992)
25. VÁGÓ, I.: *Graph Theory. Application to the Calculation of Electrical Networks*. Elsevier (1985)
26. WALTHER, H.: *Ten Applications of Graph Theory*. Kluwer (1985)
27. WEST, D.B.: *Introduction to Graph Theory*. Prentice–Hall (1996)

# Index

- across-quantity 43
- across-source 43
- across-vector 43
- acyclic directed graph 32
- adjacency matrix 34
- adjacent edges 2
- adjacent vertices 2
- admittance matrix 46
- all-vertex incidence matrix 34
- alternating path 76
- annealing algorithm 72,91
- approximation algorithm 50
- arboricity 105
- arc 27
- articulation vertex 14
- aspect 92
- augmentation 95,100
- augmenting path 76,82
- augmenting tree 77
- back edge 54,56
- basis 92
- basis exchange property 94,100
- BFS tree 59
- big-O notation 50
- binary matroid 96
- bipartite graph 17,76,97
- block 15
- bond matroid 104
- branch 21
- Breadth-First Search 59
- capacity 80
- capacity constraint 80
- chord 20
- chromatic number 89
- circuit 6,23,40,92
- circuit matrix 40
- circuit matroid 93,105
- circuit space 49
- clique 5
- closed walk 6
- cobasis 103
- cocircuit 103
- cocircuit matroid 104
- coloring of a graph 89
- complement of graph 10
- complete bipartite graph 17
- complete graph 3
- component 7,28,43
- computational complexity 50
- condensed graph 28
- connected digraph 28
- connected graph 7
- contracting of edge 13
- contraction of matroid 105
- coseparating tree 20
- cross edge 56
- cut 16
- cut matrix 36
- cut matroid 104
- cut set 16,24,36
- cut space 49
- cut vertex 14
- Davidson–Harel Algorithm 90
- decision problem 50
- degree of vertex 2
- Demoucron's Algorithm 87
- Demoucron–Malgrange–Pertuiset Algorithm 87
- dependent set 92
- Depth-First Search 53
- deterministic algorithm 50
- DFS forest 57
- DFS tree 54
- difference of graphs 11
- digraph 27
- Dijkstra's Algorithm 61
- direct sum 104
- directed edge 27
- directed graph 27
- directed spanning tree 31
- directed tree 29
- directed walk 27
- dual hereditary system 103
- dual matroid 102
- edge 1
- Edmonds Covering Theorem 104
- Edmonds–Fulkerson Theorem 104
- Edmonds–Karp Modification 84
- elimination property 95,100
- empty graph 2
- end vertex 2
- Euler's Polyhedron Formula 86
- Five-Color Theorem 89
- flow 80
- Floyd's Algorithm 63
- Ford–Fulkerson Algorithm 83
- forest 20
- forward edge 56
- Four-Color Theorem 89
- free matroid 97
- fundamental circuit 23

- fundamental circuit matrix 41
- fundamental cut set 24
- fundamental cut set matrix 39
- fundamental equations 44
- fundamental set of circuits 23
- fundamental set of cut sets 24
- graph 1
- graphic matroid 93
- greediness property 99,100
- greedy algorithm 98
- Hall's Theorem 79
- Hamiltonian circuit 61,98
- Heawood's Algorithm 90
- Heawood's Theorem 89
- hereditary family 92
- hereditary set 92
- Hopcroft–Tarjan Algorithm 87
- Hungarian Algorithm 77
- Hungarian tree 77
- impedance matrix 46
- in-degree 27
- incidence matrix 35
- independent set 92
- induced subgraph 5
- intersection of graphs 11
- intractable problem 51
- isolated vertex 2
- isomorphic graphs 18
- Jarnik's Algorithm 70
- Karp–Held Heuristics 73
- Kirchhoff's Across-Quantity Law 43
- Kirchhoff's Flow Law 80
- Kirchhoff's Through-Quantity Law 43
- Kruskal's Algorithm 67,98,104
- Kuratowski's Theorem 87
- labeled graph 18
- labeling 18
- Las Vegas algorithm 51
- leaf 29
- lightest Hamiltonian circuit 71
- lightest path 61,63
- lightest spanning tree 66
- Linear Bound 86,105
- linear matroid 96
- link 21
- loop 2,92
- Marimont's Algorithm 33
- Marriage Theorem 79
- matching 76,97
- matrix matroid 96
- matroid 100
- Matroid Covering Theorem 104
- Matroid Greediness Theorem 99
- Matroid Sum Theorem 104
- Matroid Union Theorem 104
- Max-Flow Min-Cut Theorem 83
- maximal matching 76
- maximum degree 3
- maximum matching 76,84
- minimum degree 3
- Minimum Degree Bound 87
- minor 105
- Monte Carlo algorithm 51
- multiplicity 1,12
- multiset 1
- Nas–Williams Formula 105
- $\mathcal{NP}$  51
- $\mathcal{NP}$ -complete 51,71
- $\mathcal{NP}$ -hard 51,91
- nondeterministic algorithm 50
- null graph 2
- nullity of graph 8
- open walk 6 out-degree 27
- $\mathcal{P}$  51
- parallel edges 2
- parallel elements 92
- partition matroid 97
- path 6
- pendant edge 2
- pendant vertex 2
- perfect matching 79
- planar embedding 85
- planar graph 85,104,105
- polynomial time 51
- polynomial space 51
- potential vector 43
- Prim's Algorithm 70
- probabilistic algorithm 51
- proper difference 12
- property of induced circuits 96,100
- quasi-strongly connected digraph 29
- rank function 93
- rank of graph 8
- rank of matroid 93
- reachability matrix 52
- reference vertex 35
- region 85
- removal of edge 13
- removal of vertex 12
- representation 96
- restriction of matroid 105
- ring sum of graphs 11,23
- root 29
- separable graph 14

short-circuiting of vertices 13  
 shortest path 61  
 simple graph 2  
 spanning tree 20  
 stationary linear network 43  
 stochastic algorithm 51  
 strong absorptivity 95,100  
 strongly connected 28  
 strongly connected component 28  
 subforest 20  
 subgraph 3  
 submodularity 100  
 subtree 20  
 symmetric difference 11  
 Tellegen's Theorem 48  
 through-quantity 43  
 through-source 43  
 through-vector 43  
 topological sorting 32  
 tractable problem 51  
 trail 6  
 transport network 80  
 transversal matroid 97  
 Travelling Salesman's Problem 71  
 tree 20,29  
 tree edge 54,56,59  
 trivial graph 2  
 underlying graph 27  
 uniform matroid 97  
 uniformity 94,100  
 union of graphs 11  
 union of matroids 104  
 vectorial matroid 96  
 vertex 1  
 walk 6  
 Warshall's Algorithm 52  
 weak absorptivity 94,100  
 weights 18  
 Whitney's Planarity Theorem 104  
 Whitney's Theorem 103

## Basic Concepts in Graph Theory

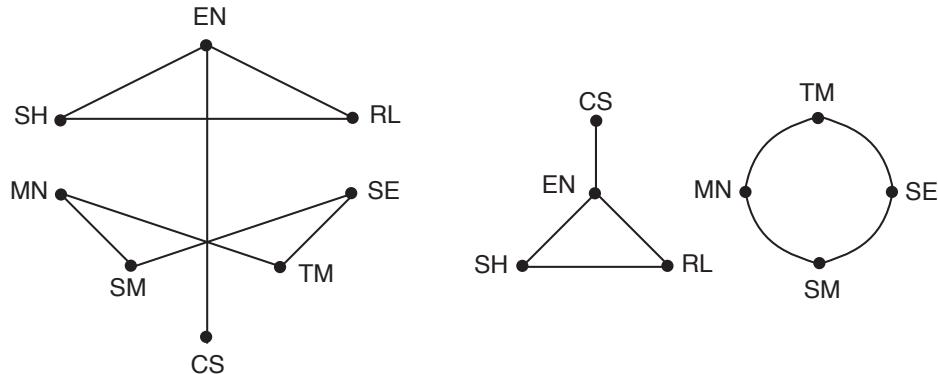
### Section 1: What is a Graph?

There are various types of graphs, each with its own definition. Unfortunately, some people apply the term “graph” rather loosely, so you can’t be sure what type of graph they’re talking about unless you ask them. After you have finished this chapter, we expect you to use the terminology carefully, not loosely. To motivate the various definitions, we’ll begin with some examples.

**Example 1 (A computer network)** Computers are often linked with one another so that they can interchange information. Given a collection of computers, we would like to describe this linkage in fairly clean terms so that we can answer questions such as “How can we send a message from computer A to computer B using the fewest possible intermediate computers?”

We could do this by making a list that consists of pairs of computers that are connected. Note that these pairs are unordered since, if computer C can communicate with computer D, then the reverse is also true. (There are sometimes exceptions to this, but they are rare and we will assume that our collection of computers does not have such an exception.) Also, note that we have implicitly assumed that the computers are distinguished from each other: It is insufficient to say that “A PC is connected to a Mac.” We must specify which PC and which Mac. Thus, each computer has a unique identifying label of some sort.

For people who like pictures rather than lists, we can put dots on a piece of paper, one for each computer. We label each dot with a computer’s identifying label and draw a curve connecting two dots if and only if the corresponding computers are connected. Note that the shape of the curve does not matter (it could be a straight line or something more complicated) because we are only interested in whether two computers are connected or not. Below are two such pictures of the same graph. Each computer has been labeled by the initials of its owner.



Computers (vertices) are indicated by dots (•) with labels. The connections (edges) are indicated by lines. When lines cross, they should be thought of as cables that lie on top of each other — not as cables that are joined. □

## Basic Concepts in Graph Theory

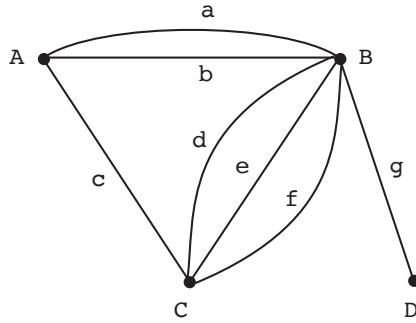
The notation  $\mathcal{P}_k(V)$  stands for the set of all  $k$ -element subsets of the set  $V$ . Based on the previous example we have

**Definition 1 (Simple graph)** A simple graph  $G$  is a pair  $G = (V, E)$  where

- $V$  is a finite set, called the vertices of  $G$ , and
- $E$  is a subset of  $\mathcal{P}_2(V)$  (i.e., a set  $E$  of two-element subsets of  $V$ ), called the edges of  $G$ .

In our example, the vertices are the computers and a pair of computers is in  $E$  if and only if they are connected.

**Example 2 (Routes between cities)** Imagine four cities named, with characteristic mathematical charm,  $A, B, C$  and  $D$ . Between these cities there are various routes of travel, denoted by  $a, b, c, d, e, f$  and  $g$ . Here is picture of this situation:



Looking at this picture, we see that there are three routes between cities  $B$  and  $C$ . These routes are named  $d, e$  and  $f$ . Our picture is intended to give us only information about the interconnections between cities. It leaves out many aspects of the situation that might be of interest to a traveler. For example, the nature of these routes (rough road, freeway, rail, etc.) is not portrayed. Furthermore, unlike a typical map, no claim is made that the picture represents in any way the distances between the cities or their geographical placement relative to each other. The object shown in this picture is called a *graph*.

Following our previous example, one is tempted to list the pairs of cities that are connected; in other words, to extract a simple graph from the information. Unfortunately, this does not describe the problem adequately because there can be more than one route connecting a pair of cities; e.g.,  $d, e$  and  $f$  connecting cities  $B$  and  $C$  in the figure. How can we deal with this? Here is a precise definition of a graph of the type required to handle this type of problem.  $\square$

**Definition 2 (Graph)** A graph is a triple  $G = (V, E, \phi)$  where

- $V$  is a finite set, called the vertices of  $G$ ,
- $E$  is a finite set, called the edges of  $G$ , and
- $\phi$  is a function with domain  $E$  and codomain  $\mathcal{P}_2(V)$ .

## Section 1: What is a Graph?

In the pictorial representation of the cities graph,  $G = (V, E, \phi)$  where

$$V = \{A, B, C, D\}, \quad E = \{a, b, c, d, e, f, g\}$$

and

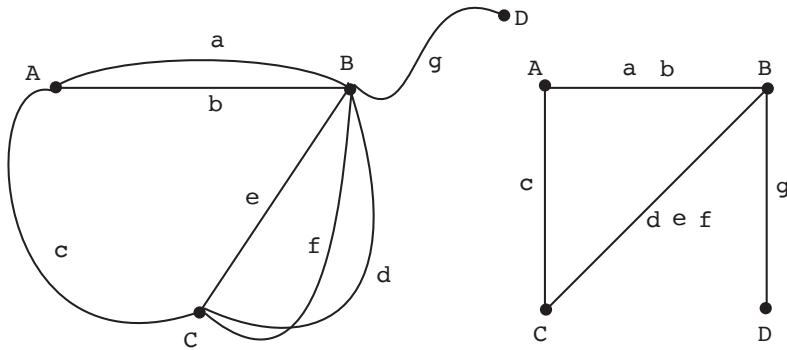
$$\phi = \begin{pmatrix} a & b & c & d & e & f & g \\ \{A, B\} & \{A, B\} & \{A, C\} & \{B, C\} & \{B, C\} & \{B, C\} & \{B, D\} \end{pmatrix}.$$

Definition 2 tells us that to specify a graph  $G$  it is necessary to specify the sets  $V$  and  $E$  and the function  $\phi$ . We have just specified  $V$  and  $\phi$  in *set theoretic* terms. The picture of the cities graph specifies the same  $V$  and  $\phi$  in pictorial terms. The set  $V$  is represented clearly by dots ( $\bullet$ ), each of which has a city name adjacent to it. Similarly, the set  $E$  is also represented clearly. The function  $\phi$  is determined from the picture by comparing the name attached to a route with the two cities connected by that route. Thus, the route name  $d$  is attached to the route with endpoints  $B$  and  $C$ . This means that  $\phi(d) = \{B, C\}$ .

Note that, since part of the definition of a function includes its codomain and domain,  $\phi$  determines  $\mathcal{P}_2(V)$  and  $E$ . Also,  $V$  can be determined from  $\mathcal{P}_2(V)$ . Consequently, we could have said that a graph is a function  $\phi$  whose domain is a finite set and whose codomain is  $\mathcal{P}_2(V)$  for some finite set  $V$ . Instead, we choose to specify  $V$  and  $E$  explicitly because the vertices and edges play a fundamental role in thinking about a graph  $G$ .

The function  $\phi$  is sometimes called the *incidence function* of the graph. The two elements of  $\phi(x) = \{u, v\}$ , for any  $x \in E$ , are called the vertices of the edge  $x$ , and we say  $u$  and  $v$  are *joined* by  $x$ . We also say that  $u$  and  $v$  are *adjacent vertices* and that  $u$  is *adjacent to*  $v$  or, equivalently,  $v$  is adjacent to  $u$ . For any  $v \in V$ , if  $v$  is a vertex of an edge  $x$  then we say  $x$  is *incident* on  $v$ . Likewise, we say  $v$  is a member of  $x$ ,  $v$  is on  $x$ , or  $v$  is in  $x$ . Of course,  $v$  is a member of  $x$  actually means  $v$  is a member of  $\phi(x)$ .

Here are two additional pictures of the same cities graph given above:



The drawings look very different but exactly the same set  $V$  and function  $\phi$  are specified in each case. It is *very important* that you understand exactly what information is needed to completely specify the graph. When thinking in terms of cities and routes between them, you naturally want the pictorial representation of the cities to represent their geographical positioning also. If the pictorial representation does this, that's fine, but it is not a part of the information required to define a graph. Geographical location is extra information. The geometrical positioning of the vertices  $A, B, C$  and  $D$  is very different, in the first of the two pictorial representations above, than it was in our original representation of the cities. However, in each of these cases, the vertices on a given edge are the same and hence the

## Basic Concepts in Graph Theory

graphs specified are the same. In the second of the two pictures above, a different method of specifying the graph is given. There,  $\phi^{-1}$ , the inverse of  $\phi$ , is given. For example,  $\phi^{-1}(\{C, B\})$  is shown to be  $\{d, e, f\}$ . Knowing  $\phi^{-1}$  determines  $\phi$  and hence determines  $G$  since the vertices  $A, B, C$  and  $D$  are also specified.

**Example 3 (Loops)** A *loop* is an edge that connects a vertex to itself. Graphs and simple graphs as defined in Definitions 1 and 2 cannot have loops. Why? Suppose  $e \in E$  is a loop in a graph that connects  $v \in V$  to itself. Then  $\phi(e) = \{v, v\} = \{v\}$  because repeated elements in the description of a set count only once — they're the same element. Since  $\{v\} \notin \mathcal{P}_2(V)$ , the range of  $\phi$ , we cannot have  $\phi(e) = \{v, v\}$ . In other words, we cannot have a loop.

Thus, if we want to allow loops, we will have to change our definitions. For a graph, we expand the codomain of  $\phi$  to be  $\mathcal{P}_2(V) \cup \mathcal{P}_1(V)$ . For a simple graph we need to change the set of allowed edges to include loops. This can be done by saying that  $E$  is a subset of  $\mathcal{P}_2(V) \cup \mathcal{P}_1(V)$  instead of a subset of just  $\mathcal{P}_2(V)$ . For example, if  $V = \{1, 2, 3\}$  and  $E = \{\{1, 2\}, \{2\}, \{2, 3\}\}$ , this simple graph has a loop at vertex 2 and vertex 2 is connected by edges to the other two vertices. When we want to allow loops, we speak of a graph with loops or a simple graph with loops.

Examples of graphs with loops appear in the exercises.  $\square$

We have two definitions, Definition 1 (simple graph) and Definition 2 (graph). How are they related? Let  $G = (V, E)$  be a simple graph. Define  $\phi: E \rightarrow E$  to be the identity map; i.e.,  $\phi(e) = e$  for all  $e \in E$ . The graph  $G' = (V, E, \phi)$  is essentially the same as  $G$ . There is one subtle difference in the pictures: The edges of  $G$  are unlabeled but each edge of  $G'$  is labeled by a set consisting of the two vertices at its ends. But this extra information is contained already in the specification of  $G$ . Thus, simple graphs are a special case of graphs.

**Definition 3 (Degrees of vertices)** Let  $G = (V, E, \phi)$  be a graph and  $v \in V$  a vertex. Define the *degree* of  $v$ ,  $d(v)$  to be the number of  $e \in E$  such that  $v \in \phi(e)$ ; i.e.,  $e$  is incident on  $v$ .

Suppose  $|V| = n$ . Let  $d_1, d_2, \dots, d_n$ , where  $d_1 \leq d_2 \leq \dots \leq d_n$  be the sequence of degrees of the vertices of  $G$ , sorted by size. We refer to this sequence as the *degree sequence* of the graph  $G$ .

In the graph for routes between cities,  $d(A) = 3$ ,  $d(B) = 6$ ,  $d(C) = 4$ , and  $d(D) = 1$ . The degree sequence is 1,3,4,6.

Sometimes we are interested only in the “structure” or “form” of a graph and not in the names (labels) of the vertices and edges. In this case we are interested in what is called an unlabeled graph. A picture of an unlabeled graph can be obtained from a picture of a graph by erasing all of the names on the vertices and edges. This concept is simple enough, but is difficult to use mathematically because the idea of a picture is not very precise.

The concept of an *equivalence relation* on a set is an important concept in mathematics and computer science. We'll explore it here and will use it to develop an intuitive

## Section 1: What is a Graph?

understanding of unlabeled graphs. Later we will use it to define connected components and biconnected components. Equivalence relations are discussed in more detail in *A Short Course in Discrete Mathematics*, the text for the course that precedes this course.

**Definition 4 (Equivalence relation)** An equivalence relation on a set  $S$  is a partition of  $S$ . We say that  $s, t \in S$  are equivalent if and only if they belong to the same block (called an equivalence class in this context) of the partition. If the symbol  $\sim$  denotes the equivalence relation, then we write  $s \sim t$  to indicate that  $s$  and  $t$  are equivalent.

**Example 4 (Equivalence relations)** Let  $S$  be any set and let all the blocks of the partition have one element. Two elements of  $S$  are equivalent if and only if they are the same. This rather trivial equivalence relation is, of course, denoted by “ $=$ ”.

Now let the set be the integers  $\mathbb{Z}$ . Let's try to define an equivalence relation by saying that  $n$  and  $k$  are equivalent if and only if they differ by a multiple of 24. Is this an equivalence relation? If it is we should be able to find the blocks of the partition. There are 24 of them, which we could number  $0, \dots, 23$ . Block  $j$  consists of all integers which equal  $j$  plus a multiple of 24; that is, they have a remainder of  $j$  when divided by 24. Since two numbers belong to the same block if and only if they both have the same remainder when divided by 24, it follows that they belong to the same block if and only if their difference gives a remainder of 0 when divided by 24, which is the same as saying their difference is a multiple of 24. Thus this partition does indeed give the desired equivalence relation.

Now let the set be  $\mathbb{Z} \times \mathbb{Z}^*$ , where  $\mathbb{Z}^*$  is the set of all integers except 0. Write  $(a, b) \sim (c, d)$  if and only if  $ad = bc$ . With a moment's reflection, you should see that this is a way to check if the two fractions  $a/b$  and  $c/d$  are equal. We can label each equivalence class with the fraction  $a/b$  that it represents. In an axiomatic development of the rationals from the integers, one defines a rational number to be just such an equivalence class and proves that it is possible to add, subtract, multiply and divide equivalence classes.

Suppose we consider all functions  $S = \underline{m^n}$ . We can define a partition of  $S$  in a number of different ways. For example, we could partition the functions  $f$  into blocks where the sum of the integers in the  $\text{Image}(f)$  is constant, where the max of the integers in  $\text{Image}(f)$  is constant, or where the “type vector” of the function, namely, the number of 1's, 2's, etc. in  $\text{Image}(f)$ , is constant. Each of these defines a partition of  $S$ .  $\square$

In the next theorem we provide necessary and sufficient conditions for an equivalence relation. Verifying the conditions is a useful way to prove that some particular situation is an equivalence relation. Recall that a *binary relation* on a set  $S$  is a subset  $R$  of  $S \times S$ .

**Theorem 1 (Equivalence Relations)** Let  $S$  be a set and suppose that we have a binary relation  $R \subseteq S \times S$ . We write  $s \sim t$  whenever  $(s, t) \in R$ . This is an equivalence relation if and only if the following three conditions hold.

- (i) (Reflexive) For all  $s \in S$  we have  $s \sim s$ .
- (ii) (Symmetric) For all  $s, t \in S$  such that  $s \sim t$  we have  $t \sim s$ .
- (iii) (Transitive) For all  $r, s, t \in S$  such that  $r \sim s$  and  $s \sim t$  we have  $r \sim t$ .

## Basic Concepts in Graph Theory

**Proof:** We first prove that an equivalence relation satisfies (i)–(iii). Suppose that  $\sim$  is an equivalence relation. Since  $s$  belongs to whatever block it is in, we have  $s \sim s$ . Since  $s \sim t$  means that  $s$  and  $t$  belong to the same block, we have  $s \sim t$  if and only if we have  $t \sim s$ . Now suppose that  $r \sim s \sim t$ . Then  $r$  and  $s$  are in the same block and  $s$  and  $t$  are in the same block. Thus  $r$  and  $t$  are in the same block and so  $r \sim t$ .

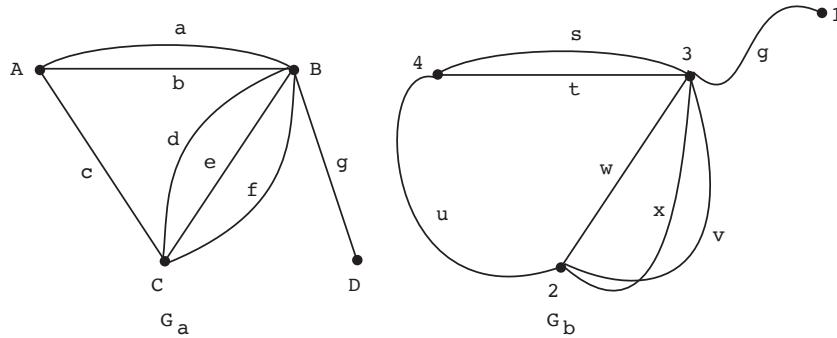
We now suppose that (i)–(iii) hold and prove that we have an equivalence relation. What would the blocks of the partition be? Everything equivalent to a given element should be in the same block. Thus, for each  $s \in S$  let  $B(s)$  be the set of all  $t \in S$  such that  $s \sim t$ . We must show that the set of these sets form a partition of  $S$ .

In order to have a partition of  $S$ , we must have

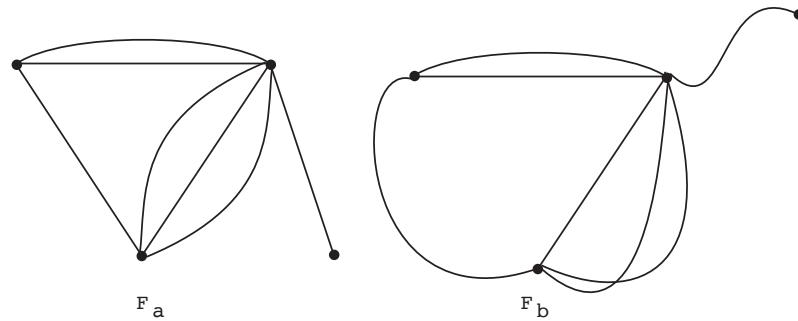
- (a) the  $B(s)$  are nonempty and every  $t \in S$  is in some  $B(s)$  and
- (b) for every  $p, q \in S$ ,  $B(p)$  and  $B(q)$  are either equal or disjoint.

Since  $\sim$  is reflexive,  $s \in B(s)$ , proving (a). Suppose  $x \in B(p) \cap B(q)$  and  $y \in B(p)$ . We have,  $p \sim x$ ,  $q \sim x$  and  $p \sim y$ . Thus  $q \sim x \sim p \sim y$  and so  $y \in B(q)$ , proving that  $B(p) \subseteq B(q)$ . Similarly  $B(q) \subseteq B(p)$  and so  $B(p) = B(q)$ . This proves (b).  $\square$

**Example 5 (Equivalent forms)** Consider the following two graphs, represented by pictures:



Now let's remove all symbols representing edges and vertices. What we have left are two “forms” on which the graphs were drawn. You can think of drawing a picture of a graph as a two step process: (1) draw the form; (2) add the labels. One student referred to these forms as “ghosts of departed graphs.” Note that form  $F_a$  and form  $F_b$  have a certain eerie similarity (appropriate for ghosts).



## Section 1: What is a Graph?

If you use your imagination a bit you can see that form  $F_b$  can be transformed into form  $F_a$  by sliding vertices around and bending, stretching, and contracting edges as needed. The edges need not be detached from their vertices in the process and edges and vertices, while being moved, can pass through each other like shadows. Let's refer to the sliding, bending, stretching, and contracting process as "morphing" form  $F_a$  into  $F_b$ . Morphing is easily seen to define an equivalence relation  $\sim$  on the set of all forms. Check out reflexive, symmetric, and transitive, for the morphing relation  $\sim$ . By Theorem 1, the morphing equivalence relation partitions the set of all forms of graphs into blocks or equivalence classes. This is a good example where it is easier to think of the relation  $\sim$  than to think globally of the partition of the forms.

Now suppose we have any two graphs,  $G_a = (V_a, E_a, \phi_a)$  and  $G_b = (V_b, E_b, \phi_b)$ . Think of these graphs not as pictures, but as specified in terms of sets and functions. Now choose forms  $F_a$  and  $F_b$  for  $G_a$  and  $G_b$  respectively, and draw their pictures. We leave it to your intuition to accept the fact that either  $F_a \sim F_b$ , no matter what you choose for  $F_a$  and  $F_b$ , or  $F_a \not\sim F_b$  no matter what your choice is for the forms  $F_a$  and  $F_b$ . If  $F_a \sim F_b$  we say that  $G_a$  and  $G_b$  are *isomorphic graphs* and write  $G_a \approx G_b$ . The fact that  $\sim$  is an equivalence relation forces  $\approx$  to be an equivalence relation also. In particular, two graphs  $G_a$  and  $G_b$  are isomorphic if and only if you can choose any form  $F_a$  for drawing  $G_a$  and use that same form for  $G_b$ .  $\square$

In general, deciding whether or not two graphs are isomorphic can be very difficult business. You can imagine how hard it would be to look at the forms of two graphs with thousands of vertices and edges and deciding whether or not those forms are equivalent. There are no general computer programs that do the task of deciding isomorphism well. For graphs known to have special features, isomorphism of graphs can sometimes be decided efficiently. In general, if someone presents you with two graphs and asks you if they are isomorphic, your best answer is "no." You will be right most of the time.

---

### \*Random Graphs

We now look briefly at a subject called *random graphs*. They often arise in the analysis of graphical algorithms and of systems which can be described graphically (such as the web). There are two basic ways to describe random graphs. One is to let the probability space be the set of all graphs with, for example,  $n$  vertices and  $q$  edges and use the uniform distribution. The other, which is often easier to study is described in the following definition. It is the one we study here.

**\*Definition 5 (Random graph model)** *Let  $\mathcal{G}(n, p)$  be the probability space obtained by letting the elementary events be the set of all  $n$ -vertex simple graphs with  $V = \underline{n}$ . If  $G \in \mathcal{G}(n, p)$  has  $m$  edges, the  $P(G) = p^m q^{N-m}$  where  $q = 1 - p$  and  $N = \binom{n}{2}$ .*

We need to show that  $\mathcal{G}(n, p)$  is a probability space. There is a nice way to see this by reinterpreting  $P$ . List the  $N = \binom{n}{2}$  vertices  $\mathcal{P}_2(V)$  in lex order. Let the sample space be  $U = \times^N \{\text{choose}, \text{reject}\}$  with  $P(a_1, \dots, a_N) = P^*(a_1) \times \dots \times P^*(a_N)$  where  $P^*(\text{choose}) = p$

## Basic Concepts in Graph Theory

and  $P^*(\text{reject}) = 1 - p$ . We've met this before in Unit Fn and seen that it is a probability space. To see that it is, note that  $P \geq 0$  and

$$\begin{aligned} \sum_{a_1, \dots, a_N} P(a_1, \dots, a_N) &= \sum_{a_1, \dots, a_N} P^*(a_1) \times \dots \times P^*(a_N) \\ &= \left( \sum_{a_1} P^*(a_1) \right) \times \dots \times \left( \sum_{a_N} P^*(a_N) \right) \\ &= (p + (1 - p)) \times \dots \times (p + (1 - p)) = 1 \times \dots \times 1 = 1. \end{aligned}$$

Why is this the same as the definition? Think of the chosen pairs as the edges of a graph chosen randomly from  $\mathcal{G}(n, p)$ . If  $G$  has  $m$  edges, then its probability should be  $p^m(1-p)^{N-m}$  according to the definition. On the other hand, since  $G$  has  $m$  edges, exactly  $m$  of  $a_1, \dots, a_N$  equal “choose” and so, in the new space,  $P(a_1, \dots, a_N) = p^m(1-p)^{N-m}$  also. We say that we are choosing the edges of the random graph independently.

**\*Example 6 (The number of edges in random graph)** Let  $X$  be a random variable that counts the number of edges in a random graph in  $\mathcal{G}(n, p)$ . What are the expected value and variance of  $X$ ? In  $U = \times^N \{\text{choose, reject}\}$ , let

$$X_i(a_1, \dots, a_N) = \begin{cases} 1 & \text{if } a_i = \text{choose,} \\ 0 & \text{if } a_i = \text{reject.} \end{cases}$$

You should be able to see that  $X = X_1 + \dots + X_N$  and that the  $X_i$  are independent random variables with  $P(X_i = 1) = p$ . This is just the binomial distribution (Unit Fn). We showed that the mean is  $Np$  and the variance is  $Npq$ , where  $N = \binom{n}{2}$  and  $q = 1 - p$ .  $\square$

**\*Example 7 (Triangles in random graphs)** How often can we find 3 vertices  $\{u, v, w\}$  in a random graph so that  $\{u, v\}$ ,  $\{u, w\}$ , and  $\{v, w\}$  are all edges of the graph? In other words, how often can we find a “triangle”? How can we do this?

First, we need a sample space. It will be the random graph space introduced in Definition 5. Since we want to count something (triangles), we need a random variable. Let  $X$  be a random variable whose value is the number of triples of vertices such that the three possible edges connecting them are present in the random graph. In other words,  $X$  is defined for each graph,  $G$ , and its value,  $X(G)$ , is the number of triangles in the graph  $G$ . We want to compute  $E(X)$ . It would also be nice to compute  $\text{Var}(X)$  since that gives us some idea of how much  $X$  tends to vary from graph to graph — large  $\text{Var}(X)$  means there tends to be a lot of variation in the number of triangles from graph to graph and small  $\text{Var}(X)$  means there tends to be little variation.

Let  $X_{u,v,w}$  be a random variable which is 1 if the triangle with vertices  $\{u, v, w\}$  is present and 0 otherwise. Then  $X$  is the sum of  $X_{u,v,w}$  over all  $\{u, v, w\} \in \mathcal{P}_3(V)$ . Since expectation is linear,  $E(X)$  is the sum of  $E(X_{u,v,w})$  over all  $\{u, v, w\} \in \mathcal{P}_3(V)$ . Clearly  $E(X_{u,v,w})$  does not depend on the particular triple. Since there are  $\binom{n}{3}$  possibilities for  $\{u, v, w\}$ ,  $E(X) = \binom{n}{3}E(X_{1,2,3})$ .

We want to compute  $E(X_{1,2,3})$ . It is given by

$$E(X_{1,2,3}) = 0P(X_{1,2,3} = 0) + 1P(X_{1,2,3} = 1) = P(X_{1,2,3} = 1).$$

## Section 1: What is a Graph?

The only way  $X_{1,2,3} = 1$  can happen is for the edges  $\{1, 2\}$ ,  $\{1, 3\}$ , and  $\{2, 3\}$  to all be present in the graph. (We don't care about any of the other possible edges.) Since each of these events has probability  $p$  and the events are independent we have  $P(X_{1,2,3} = 1) = p^3$ . Thus  $E(X_{1,2,3}) = p^3$  and so  $E(X) = \binom{n}{3}p^3$ . In other words, on average we see about  $\binom{n}{3}p^3$  triangles. For example, if  $p = 1/2$  all graphs are equally likely (You should show this.) and so the average number of triangles over all graphs with  $n$  vertices is  $\binom{n}{3}/8$ . When  $n = 5$ , this average is 1.25. Can you verify this by looking at all the 5-vertex graphs? How much work is involved?

What happens when  $n$  is very large? Then  $\binom{n}{3} = \frac{n(n-1)(n-2)}{6}$  “behaves like”  $n^3/6$ . (“Behaves like” means that, as  $n$  goes to infinity, the limit of the ratio  $\binom{n}{3}/(n^3/6)$  is 1.) Thus the expected number of triangles behaves like  $(np)^3/6$ .

What about the variance? We'll work it out in the next example. For now, we'll simply tell you that it behaves like  $n^4p^3(1 - p^2)/2$ . What does this tell us for large  $n$ ? The standard deviation behaves like  $n^2p^{3/2}\sqrt{(1 - p^2)/2}$ . A more general version of the central limit theorem than we have discussed tells us the number of triangles tends to have a normal distribution with  $\mu = (np)^3/6$  and  $\sigma = n^2p^{3/2}\sqrt{(1 - p^2)/2}$ . If  $p$  is constant,  $\sigma$  will grow like a constant times  $n^2$ , which is much smaller than  $\mu$  for large  $n$ . Thus the number of triangles in a random graph is almost surely close to  $(np)^3/6$ .  $\square$

**\*Example 8 (Variance for triangles in random graphs)** This is a continuation of the previous example. Since the various  $X_{u,v,w}$  may not be independent, this is harder. Since  $\text{Var}(X) = E(X^2) - E(X)^2$ , we will compute  $E(X^2)$ . Since  $X$  is a sum of terms of the form  $X_{r,s,t}$ ,  $X^2$  is a sum of terms of the form  $X_{u,v,w}X_{a,b,c}$ . Using linearity of expectation, we need to compute  $E(X_{u,v,w}X_{a,b,c})$  for each possibility and add them up.

Now for the tricky part: This expectation depends on how many vertices  $\{u, v, w\}$  and  $\{a, b, c\}$  have in common.

- If  $\{u, v, w\} = \{a, b, c\}$ , then  $X_{u,v,w}X_{a,b,c} = X_{u,v,w}$  and its expectation is  $p^3$  by the previous example.
- If  $\{u, v, w\}$  and  $\{a, b, c\}$  have two vertices in common, then the two triangles have only 5 edges total because they have a common edge. Note that  $X_{u,v,w}X_{a,b,c}$  is 1 if all five edges are present and is 0 otherwise. Reasoning as in the previous example, the expectation is  $p^5$ .
- If  $\{u, v, w\}$  and  $\{a, b, c\}$  have less than two vertices in common, we are concerned about six edges and obtain  $p^6$  for the expectation.

To add up the results in the previous paragraph, we need to know how often each occurs in

$$X^2 = \left( \sum_{\{u,v,w\} \in \mathcal{P}_3(V)} X_{u,v,w} \right) \left( \sum_{\{a,b,c\} \in \mathcal{P}_3(V)} X_{a,b,c} \right) = \sum_{\substack{\{u,v,w\} \in \mathcal{P}_3(V) \\ \{a,b,c\} \in \mathcal{P}_3(V)}} X_{u,v,w}X_{a,b,c}.$$

- When  $\{u, v, w\} = \{a, b, c\}$ , we are only free to choose  $\{u, v, w\}$  and this can be done in  $\binom{n}{3}$  ways so we have  $\binom{n}{3}p^3$  contributed to  $E(X^2)$ .

## Basic Concepts in Graph Theory

- Suppose  $\{u, v, w\}$  and  $\{a, b, c\}$  have two vertices in common. How many ways can this happen? We can first choose  $\{u, v, w\}$ . Then choose two of  $u, v, w$  to be in  $\{a, b, c\}$  and then choose the third vertex in  $\{a, b, c\}$  to be different from  $u, v$ , and  $w$ . This can be done in

$$\binom{n}{3} \times \binom{3}{2} \times (n-3) = 3(n-3) \binom{n}{3} = 12 \binom{n}{4}$$

ways. Multiplying this by  $p^5$  gives its contribution to  $E(X^2)$ .

- The remaining case, one vertex or no vertices in common, can be done in a similar fashion. Alternatively, we can simply subtract the above counts from all possible ways of choosing  $\{u, v, w\}$  and  $\{a, b, c\}$ . This gives us

$$\binom{n}{3} \times \binom{n}{3} - \binom{n}{3} - 12 \binom{n}{4}$$

for the third case. Multiplying this by  $p^6$  gives its contribution to  $E(X^2)$ .

Since  $E(X)^2 = \binom{n}{3}^2 p^6$ , we have that

$$\text{Var}(X) = E(X)^2 - E(X^2) = \binom{n}{3} (p^3 - p^6) + 12 \binom{n}{4} (p^3 - p^5),$$

after a bit of algebra using the results in the preceding paragraph. Whew!  $\square$

The previous material would be more difficult if we had used the model for random graphs that was suggested before Definition 5. Why is this? The model we are using lets us ignore possible edges that we don't care about. The other model does not because we must be sure that the total number of edges is correct.

---

## Exercises for Section 1

- 1.1.** We are interested in the number of simple graphs with  $V = \underline{n}$ .

- (a) Prove that there are  $2^{\binom{n}{2}}$  (2 to the power  $\binom{n}{2}$ ) such simple graphs.
- (b) How many of them have exactly  $q$  edges?

- 1.2.** Let  $(V, E, \phi)$  be a graph and let  $d(v)$  be the degree of the vertex  $v \in V$ . Prove that  $\sum_{v \in V} d(v) = 2|E|$ , an even number. Conclude that the number of vertices  $v$  for which  $d(v)$  is odd is even.

- 1.3.** Let  $Q = (V, E, \phi)$  be a graph where

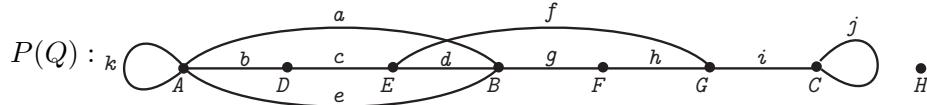
$$V = \{A, B, C, D, E, F, G, H\}, \quad E = \{a, b, c, d, e, f, g, h, i, j, k\}$$

## Section 1: What is a Graph?

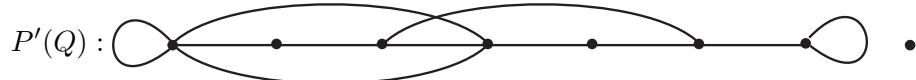
and

$$\phi = \begin{pmatrix} a & b & c & d & e & f & g & h & i & j & k \\ A & A & D & E & A & E & B & F & G & C & A \\ B & D & E & B & B & G & F & G & C & C & A \end{pmatrix}.$$

In this representation of  $\phi$ , the first row specifies the edges and the two vertices below each edge specify the vertices incident on that edge. Here is a pictorial representation  $P(Q)$  of this graph.



Note that  $\phi(k) = \{A, A\} = \{A\}$ . Such an edge is called a *loop*. (See Example 3.) Adding a loop to a vertex increases its degree by two. The vertex  $H$ , which does not belong to  $\phi(x)$  for any edge  $x$  (i.e., has no edge incident upon it), is called an *isolated vertex*. The degree of an isolated vertex is zero. Edges, such as  $a$  and  $e$  of  $Q$ , with the property that  $\phi(a) = \phi(e)$  are called *parallel edges*. If all edge and vertex labels are removed from  $P(Q)$  then we get the following picture  $P'(Q)$ :



The picture  $P'(Q)$  represents the “form” of the graph just described and is sometimes referred to as a pictorial representation of the “unlabeled” graph associated with  $Q$ . (See Example 5.) For each of the following graphs  $R$ , where  $R = (V, E, \phi)$ ,  $V = \{A, B, C, D, E, F, G, H\}$ , draw a pictorial representation of  $R$  by starting with  $P'(Q)$  removing and/or adding as few edges as possible and then labeling the resulting picture with the edges and vertices of  $R$ . A graph  $R$  which require no additions or removals of edges is said to be “of the same form as” or “isomorphic to” the graph  $Q$  (Example 5).

(a) Let

$$E = \{a, b, c, d, e, f, g, h, i, j, k\}$$

be the set of edges of  $R$  and

$$\phi = \begin{pmatrix} a & b & c & d & e & f & g & h & i & j & k \\ C & C & F & A & H & E & E & A & D & A & A \\ C & G & G & H & H & H & F & H & G & D & F \end{pmatrix}.$$

(b) Let

$$E = \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11\}$$

be the set of edges of  $R$  and

$$\phi = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 \\ A & E & E & E & F & G & H & B & C & D & E \\ G & H & E & F & G & H & B & C & D & D & H \end{pmatrix}.$$

## Basic Concepts in Graph Theory

**1.4.** Let  $Q = (V, E, \phi)$  be the graph where

$$V = \{A, B, C, D, E, F, G, H\}, \quad E = \{a, b, c, d, e, f, g, h, i, j, k, l\}$$

and

$$\phi = \begin{pmatrix} a & b & c & d & e & f & g & h & i & j & k & l \\ A & A & D & E & A & E & B & F & G & C & A & E \\ B & D & E & B & B & G & F & G & C & C & A & G \end{pmatrix}.$$

(a) What is the degree sequence of  $Q$ ?

Consider the following unlabeled pictorial representation of  $Q$



- (a) Create a pictorial representation of  $Q$  by labeling  $P'(Q)$  with the edges and vertices of  $Q$ .
- (b) A necessary condition that a pictorial representation of a graph  $R$  can be created by labeling  $P'(Q)$  with the vertices and edges of  $R$  is that the degree sequence of  $R$  be  $(0, 2, 2, 3, 4, 4, 4, 5)$ . True or false? Explain.
- (c) A sufficient condition that a pictorial representation of a graph  $R$  can be created by labeling  $P'(Q)$  with the vertices and edges of  $R$  is that the degree sequence of  $R$  be  $(0, 2, 2, 3, 4, 4, 4, 5)$ . True or false? Explain.

**1.5.** In each of the following problems information about the degree sequence of a graph is given. In each case, decide if a graph satisfying the specified conditions exists or not. Give reasons in each case.

- (a) A graph  $Q$  with degree sequence  $(1, 1, 2, 3, 3, 5)$ ?
- (b) A graph  $Q$  with degree sequence  $(1, 2, 2, 3, 3, 5)$ , loops and parallel edges allowed?
- (c) A graph  $Q$  with degree sequence  $(1, 2, 2, 3, 3, 5)$ , no loops but parallel edges allowed?
- (d) A graph  $Q$  with degree sequence  $(1, 2, 2, 3, 3, 5)$ , no loops or parallel edges allowed?
- (e) A simple graph  $Q$  with degree sequence  $(3, 3, 3, 3)$ ?
- (f) A graph  $Q$  with degree sequence  $(3, 3, 3, 3)$ , no loops or parallel edges allowed?
- (g) A graph  $Q$  with degree sequence  $(3, 3, 3, 5)$ , no loops or parallel edges allowed?
- (h) A graph  $Q$  with degree sequence  $(4, 4, 4, 4, 4)$ , no loops or parallel edges allowed?
- (i) A graph  $Q$  with degree sequence  $(4, 4, 4, 4, 6)$ , no loops or parallel edges allowed?

**1.6.** Divide the following graphs into isomorphism equivalence classes and justify your answer; i.e., explain why you have the classes that you do. In all cases  $V = \underline{4}$ .

## Section 2: Digraphs, Paths, and Subgraphs

$$(a) \phi = \begin{pmatrix} a & b & c & d & e & f \\ \{1, 2\} & \{1, 2\} & \{2, 3\} & \{3, 4\} & \{1, 4\} & \{2, 4\} \end{pmatrix}$$

$$(b) \phi = \begin{pmatrix} A & B & C & D & E & F \\ \{1, 2\} & \{1, 4\} & \{1, 4\} & \{1, 2\} & \{2, 3\} & \{3, 4\} \end{pmatrix}$$

$$(c) \phi = \begin{pmatrix} u & v & w & x & y & z \\ \{2, 3\} & \{1, 3\} & \{3, 4\} & \{1, 4\} & \{1, 2\} & \{1, 2\} \end{pmatrix}$$

$$(d) \phi = \begin{pmatrix} P & Q & R & S & T & U \\ \{3, 4\} & \{2, 4\} & \{1, 3\} & \{3, 4\} & \{1, 2\} & \{1, 2\} \end{pmatrix}$$

**\*1.7.** In Example 7, suppose that  $p$  is a function of  $n$ , say  $p = p(n)$ .

- (a) Show that the expected number of triangles is behaves like 1 for large  $n$  if  $p(n) = 6^{1/3}/n$ .
- (b) Suppose the expected number of triangles behaves like 1. How does the expected number of edges behave?

**\*1.8.** Instead of looking for triangles as in Example 7, let's look for quadrilaterals having both diagonals. In other words, we'll look for sets of four vertices such that all of the  $\binom{4}{2} = 6$  possible edges between them are present.

- (a) Show that the expected number of such quadrilaterals is  $\binom{n}{4}p^6$ .
- (b) Suppose  $n$  is large and  $p$  is a function of  $n$  so that we expect to see 1 quadrilateral on average. About how many edges do we expect to see?
- (c) Generalize this problem from sets of 4 vertices to sets of  $k$  vertices.

**\*1.9.** Show that the variance of  $X$ , the number of triangles in a random graph as computed in Example 8 satisfies

$$\text{Var}(X) = \binom{n}{3}p^3((1-p^3) + 3(n-3)(1-p^2)) < 3n\binom{n}{3}p^3(1-p^2).$$

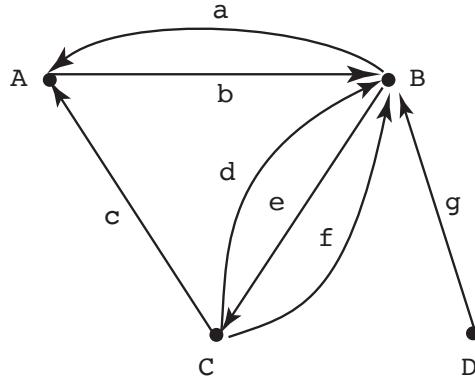
*Hint:*  $1 - p^3 < 1 - p^2 < 1$ .

## Section 2: Digraphs, Paths, and Subgraphs

In this section we introduce the notion of a directed graph and give precise definitions of some very important special substructures of both graphs and directed graphs.

## Basic Concepts in Graph Theory

**Example 9 (Flow of commodities)** Look again at Example 2. Imagine now that the symbols  $a, b, c, d, e, f$  and  $g$ , instead of standing for route names, stand for commodities (applesauce, bread, computers, etc.) that are produced in one town and shipped to another town. In order to get a picture of the flow of commodities, we need to know the directions in which they are shipped. This information is provided by picture below:



In set-theoretic terms, the information needed to construct the above picture can be specified by giving a pair  $D = (V, E, \phi)$  where  $\phi$  is a function. The domain of the function  $\phi$  is  $E = \{a, b, c, d, e, f, g\}$  and the codomain is  $V \times V$ . Specifically,

$$\phi = \begin{pmatrix} a & b & c & d & e & f & g \\ (B, A) & (A, B) & (C, A) & (C, B) & (B, C) & (C, B) & (D, B) \end{pmatrix}.$$

The structure specified by this information is an example of a *directed graph*, which we now define.  $\square$

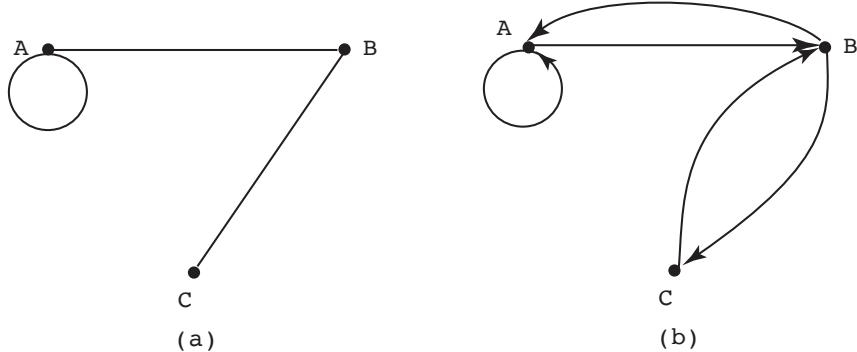
**Definition 6 (Directed graph)** A *directed graph* (or *digraph*) is a triple  $D = (V, E, \phi)$  where  $V$  and  $E$  are finite sets and  $\phi$  is a function with domain  $E$  and codomain  $V \times V$ . We call  $E$  the set of edges of the digraph  $D$  and call  $V$  the set of vertices of  $D$ .

Just as with graphs, we can define a notion of a *simple digraph*. A simple digraph is a pair  $D = (V, E)$ , where  $V$  is a set, the vertex set, and  $E \subseteq V \times V$  is the edge set. Just as with simple graphs and graphs, simple digraphs are a special case of digraphs in which  $\phi$  is the identity function on  $E$ ; that is,  $\phi(e) = e$  for all  $e \in E$ .

There is a correspondence between simple graphs and simple digraphs that is fairly common in applications of graph theory. To interpret simple graphs in terms of simple digraphs, it is best to consider simple graphs with loops (see Example 3 and Exercises for Section 1). Thus consider  $G = (V, E)$  where  $E \subseteq \mathcal{P}_2(V) \cup \mathcal{P}_1(V)$ . We can identify  $\{u, v\} \in \mathcal{P}_2(V) \cup \mathcal{P}_1(V)$  with  $(u, v) \in V \times V$  and with  $(v, u) \in V \times V$ . In the case where we have a loop,  $u = v$ , then we identify  $\{u\}$  with  $(u, u)$ . Here is a picture of a simple graph

## Section 2: Digraphs, Paths, and Subgraphs

and its corresponding digraph:

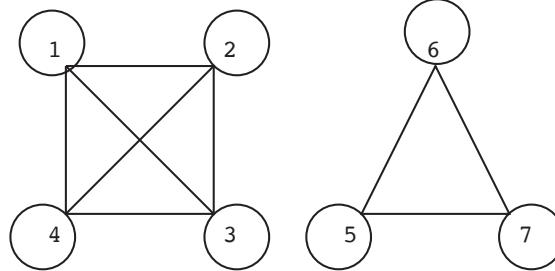


Each edge that is not a loop in the simple graph is replaced by two edges “in opposite directions” in the corresponding simple digraph. A *loop* is replaced by a *directed loop* (e.g.,  $\{A\}$  is replaced by  $(A, A)$ ).

Simple digraphs appear in mathematics under another important guise: *binary relations*. A binary relation on a set  $V$  is simply a subset of  $V \times V$ . Often the name of the relation and the subset are the same. Thus we speak of the binary relation  $E \subseteq V \times V$ . If you have absorbed all the terminology, you should be able to see immediately that  $(V, E)$  is a simple digraph and that any simple digraph  $(V', E')$  corresponds to a binary relation  $E' \subseteq V' \times V'$ .

Recall that a binary relation  $R$  is called *symmetric* if  $(u, v) \in R$  implies  $(v, u) \in R$ . Thus simple graphs with loops correspond to symmetric binary relations.

An equivalence relation on a set  $S$  is a particular type of binary relation  $R \subseteq S \times S$ . For an equivalence relation, we have  $(x, y) \in R$  if and only if  $x$  and  $y$  are equivalent (i.e., belong to the same equivalence class or block). Note that this is a symmetric relationship, so we may regard the associated simple digraph as a simple graph. Which simple graphs (with loops allowed) correspond to equivalence relations? As an example, take  $S = \underline{7}$  and take the equivalence class partition to be  $\{\{1, 2, 3, 4\}, \{5, 6, 7\}\}$ . Since everything in each block is related to everything else, there are  $\binom{4}{2} = 6$  non-loops and  $\binom{4}{1} = 4$  loops associated with the block  $\{1, 2, 3, 4\}$  for a total of ten edges. With the block  $\{5, 6, 7\}$  there are three loops and three non-loops for a total of six edges. Here is the graph of this equivalence relation:



A *complete simple graph*  $G=(V,E)$  with loops is a graph with every possible edge. That is,  $E = \mathcal{P}_2(V) \cup \mathcal{P}_1(V)$ . In the above graph, each block of the equivalence relation is replaced by the complete simple graph with loops on that block. This is the general rule.

A basic method for studying graphs and digraphs is to study substructures of these objects and their properties. One of the most important of these substructures is called a *path*.

## Basic Concepts in Graph Theory

**Definition 7 (Path, trail, walk and vertex sequence)** Let  $G = (V, E, \phi)$  be a graph.

Let  $e_1, e_2, \dots, e_{n-1}$  be a sequence of elements of  $E$  (edges of  $G$ ) for which there is a sequence  $a_1, a_2, \dots, a_n$  of distinct elements of  $V$  (vertices of  $G$ ) such that  $\phi(e_i) = \{a_i, a_{i+1}\}$  for  $i = 1, 2, \dots, n - 1$ . The sequence of edges  $e_1, e_2, \dots, e_{n-1}$  is called a path in  $G$ . The sequence of vertices  $a_1, a_2, \dots, a_n$  is called the vertex sequence of the path. (Note that since the vertices are distinct, so are the edges.)

If we require that  $e_1, \dots, e_{n-1}$  be distinct, but not that  $a_1, \dots, a_n$  be distinct, the sequence of edges is called a trail.

If we do not even require that the edges be distinct, it is called a walk.

If  $G = (V, E, \phi)$  is a directed graph, then  $\phi(e_i) = \{a_i, a_{i+1}\}$  is replaced by  $\phi(e_i) = (a_i, a_{i+1})$  in the above definition to obtain a directed path, trail, and walk respectively.

Note that the definition of a path requires that it not intersect itself (i.e., have repeated vertices), while a trail may intersect itself. Although a trail may intersect itself, it may not have repeated edges, but a walk may. If  $P = (e_1, \dots, e_{n-1})$  is a path in  $G = (V, E, \phi)$  with vertex sequence  $a_1, \dots, a_n$  then we say that  $P$  is a path from  $a_1$  to  $a_n$ . Similarly for a trail or a walk.

In the graph of Example 2, the sequence  $c, d, g$  is a path with vertex sequence  $A, C, B, D$ . If the graph is of the form  $G = (V, E)$  with  $E \subseteq \mathcal{P}_2(V)$ , then the vertex sequence alone specifies the sequence of edges and hence the path. Thus, Example 1, the vertex sequence MN, SM, SE, TM specifies the path  $\{MN, SM\}, \{SM, SE\}, \{SE, TM\}$ . Similarly for digraphs. Consider the graph of Example 9. The edge sequence  $P = (g, e, c)$  is a directed path with vertex sequence  $(D, B, C, A)$ . The edge sequence  $P = (g, e, c, b, a)$  is a directed trail, but not a directed path. The edge sequence  $P = (d, e, d)$  is a directed walk, but not a directed trail.

Note that every path is a trail and every trail is a walk, but not conversely. However, we can show that, if there is a walk between two vertices, then there is a path. This rather obvious result can be useful in proving theorems, so we state it as a theorem.

**Theorem 2 (Walk implies path)** Suppose  $u \neq v$  are vertices in the graph  $G = (V, E, \phi)$ . The following are equivalent:

- There is a walk from  $u$  to  $v$ .
- There is a trail from  $u$  to  $v$ .
- There is a path from  $u$  to  $v$ .

Furthermore, given a walk from  $u$  to  $v$ , there is a path from  $u$  to  $v$  all of whose edges are in the walk.

**Proof:** Since every path is a trail, (c) implies (b). Since every trail is a walk, (b) implies (a). Thus it suffices to prove that (a) implies (c). Let  $e_1, e_2, \dots, e_k$  be a walk from  $u$  to  $v$ . We use induction on  $n$ , the number of repeated vertices in a walk. If the walk has no repeated vertices, it is a path. This starts the induction at  $n = 0$ . Suppose  $n > 0$ . Let  $r$  be a repeated vertex. Suppose it first appears in edge  $e_i$  and last appears in edge  $e_j$ .

## Section 2: Digraphs, Paths, and Subgraphs

If  $r = u$ , then  $e_j, \dots, e_k$  is a walk from  $u$  to  $v$  in which  $r$  is not a repeated vertex. If  $r = v$ , then  $e_1, \dots, e_i$  is a walk from  $u$  to  $v$  in which  $r$  is not a repeated vertex. Otherwise,  $e_1, \dots, e_i, e_j, \dots, e_k$  is a walk from  $u$  to  $v$  in which  $r$  is not a repeated vertex. Hence there are less than  $n$  repeated vertices in this walk from  $u$  to  $v$  and so there is a path by induction. Since we constructed the path by removing edges from the walk, the last statement in the theorem follows.  $\square$

Note that the theorem and proof are valid if graph is replaced by digraph and walk, trail, and path are replaced by directed walk, trail, and path.

Another basic notion is that of a subgraph of  $G = (V, E, \phi)$ , which we will soon define. First we need some terminology about functions. By a *restriction*  $\phi'$  of  $\phi$  to  $E' \subseteq E$ , we mean the function  $\phi'$  with domain  $E'$  and satisfying  $\phi'(x) = \phi(x)$  for all  $x \in E'$ . (When forming a restriction, we may change the codomain. Of course, the new codomain must contain  $\text{Image}(\phi') = \phi(E)$ . In the following definition, the codomain of  $\phi'$  must be  $\mathcal{P}_2(V')$  since  $G'$  is required to be a graph.)

**Definition 8 (Subgraph)** Let  $G = (V, E, \phi)$  be a graph. A graph  $G' = (V', E', \phi')$  is a *subgraph* of  $G$  if  $V' \subseteq V$ ,  $E' \subseteq E$ , and  $\phi'$  is the restriction of  $\phi$  to  $E'$ .

As we have noted, the fact that  $G'$  is itself a graph means that  $\phi(x) \in \mathcal{P}_2(V')$  for each  $x \in E'$  and, in fact, the codomain of  $\phi'$  must be  $\mathcal{P}_2(V')$ . If  $G$  is a graph with loops, the codomain of  $\phi'$  must be  $\mathcal{P}_2(V') \cup \mathcal{P}_1(V')$ . This definition works equally well if  $G$  is a digraph. In that case, the codomain of  $\phi'$  must be  $V \times V$ .

**Example 10 (Subgraph — key information)** For the graph  $G = (V, E, \phi)$  below, let  $G' = (V', E', \phi')$  be defined by  $V' = \{A, B, C\}$ ,  $E' = \{a, b, c, f\}$ , and by  $\phi'$  being the restriction of  $\phi$  to  $E'$  with codomain  $\mathcal{P}_2(V')$ . Notice that  $\phi'$  is determined completely from knowing  $V'$ ,  $E'$  and  $\phi$ . Thus, to specify a subgraph  $G'$ , the key information is  $V'$  and  $E'$ .

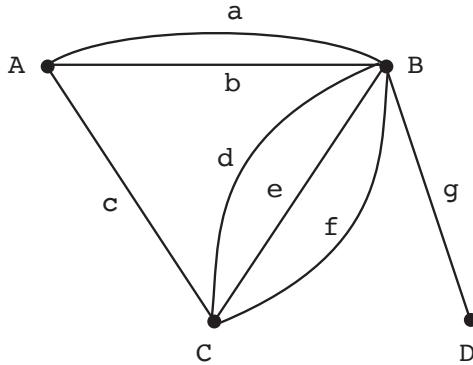
As another example from the same graph, we let  $V' = V$  and  $E' = \{a, b, c, f\}$ . In this case, the vertex  $D$  is not a member of any edge of the subgraph. Such a vertex is called an *isolated vertex* of  $G'$ . (See also Exercises for Section 1.)

One way of specifying a subgraph is to give a set of edges  $E' \subseteq E$  and take  $V'$  to be the set of all vertices on some edge of  $E'$ . In other words,  $V'$  is the union of the sets  $\phi(x)$  over all  $x \in E'$ . Such a subgraph is called the *subgraph induced by the edge set*  $E'$  or the *edge induced subgraph* of  $E'$ . The first subgraph of this example is the subgraph induced by  $E' = \{a, b, c, f\}$ .

Likewise, given a set  $V' \subseteq V$ , we can take  $E'$  to be the set of all edges  $x \in E$  such that  $\phi(x) \subseteq V'$ . The resulting subgraph is called the *subgraph induced by*  $V'$  or the *vertex induced subgraph* of  $V'$ . Referring to the picture again, the edges of the subgraph induced

## Basic Concepts in Graph Theory

by  $V' = \{C, B\}$ , are  $E' = \{d, e, f\}$ .



Look again at the above graph. In particular, consider the path  $c, a$  with vertex sequence  $C, A, B$ . Notice that the edge  $d$  has  $\phi(d) = \{C, B\}$ . The subgraph  $G' = (V', E', \phi')$ , where  $V' = \{C, A, B\}$  and  $E' = \{c, a, d\}$  is called a *cycle* of  $G$ . In general, whenever there is a path in  $G$ , say  $e_1, \dots, e_{n-1}$  with vertex sequence  $a_1, \dots, a_n$ , and an edge  $x$  with  $\phi(x) = \{a_1, a_n\}$ , then the subgraph induced by the edges  $e_1, \dots, e_{n-1}, x$  is called a *cycle* of  $G$ . Parallel edges like  $a$  and  $b$  in the preceding figure induce a cycle. A loop also induces a cycle.  $\square$

The formal definition of a cycle is:

**Definition 9 (Circuit and Cycle)** Let  $G = (V, E, \phi)$  be a graph and let  $e_1, \dots, e_n$  be a trail with vertex sequence  $a_1, \dots, a_n, a_1$ . (It returns to its starting point.) The subgraph  $G'$  of  $G$  induced by the set of edges  $\{e_1, \dots, e_n\}$  is called a *circuit* of  $G$ . The *length* of the circuit is  $n$ .

- If the only repeated vertices on the trail are  $a_1$  (the start and end), then the circuit is called a *simple circuit* or *cycle*.
- If “trail” is replaced by directed trail, we obtain a *directed circuit* and a *directed cycle*.

In our definitions, a path is a *sequence* of edges but a cycle is a *subgraph* of  $G$ . In actual practice, people often think of a cycle as a path, except that it starts and ends at the same vertex. This sloppiness rarely causes trouble, but can lead to problems in formal proofs. Cycles are closely related to the existence of multiple paths between vertices:

**Theorem 3 (Cycles and multiple paths)** Two vertices  $u \neq v$  are on a cycle of  $G$  if and only if there are at least two paths from  $u$  to  $v$  that have no vertices in common except the endpoints  $u$  and  $v$ .

**Proof:** Suppose  $u$  and  $v$  are on a cycle. Follow the cycle from  $u$  to  $v$  to obtain one path. Then follow the cycle from  $v$  to  $u$  to obtain another. Since a cycle has no repeated vertices, the only vertices that lie in both paths are  $u$  and  $v$ . On the other hand, a path from  $u$  to  $v$  followed by a path from  $v$  to  $u$  is a cycle if the paths have no vertices in common other than  $u$  and  $v$ .  $\square$

## Section 2: Digraphs, Paths, and Subgraphs

One important feature of a graph is whether or not any pair of vertices can be connected by a path. You can probably imagine, without much difficulty, applications of graph theory where this sort of “connectivity” is important. Not the least of such examples would be communication networks. Here is a formal definition of *connected* graphs.

**Definition 10 (Connected graph)** *Let  $G = (V, E, \phi)$  be a graph. If for any two distinct elements  $u$  and  $v$  of  $V$  there is a path  $P$  from  $u$  to  $v$  then  $G$  is a connected graph. If  $|V| = 1$ , then  $G$  is connected.*

We make two observations about the definition.

- Because of Theorem 2, we can replace “path” in the definition by “walk” or “trail” if we wish. (This observation is used in the next example.)
- The last sentence in the definition is not really needed. To see this, suppose  $|V| = 1$ . Now  $G$  is connected if, for any two *distinct* elements  $u$  and  $v$  of  $V$  there is a path from  $u$  to  $v$ . This is trivially satisfied since we cannot find two distinct elements in the one element set  $V$ .

The graph of Example 1 has two distinct “pieces.” It is not a connected graph. There is, for example, no path from  $u = TM$  to  $v = CS$ . Note that one piece of this graph consists of the vertex induced subgraph of the vertex set  $\{CS, EN, SH, RL\}$  and the other piece consists of the vertex induced subgraph of  $\{TM, SE, MN, SM\}$ . These pieces are called *connected components* of the graph. This is the case in general for a graph  $G = (V, E, \phi)$ : The vertex set is partitioned into subsets  $V_1, V_2, \dots, V_m$  such that if  $u$  and  $v$  are in the same subset then there is a path from  $u$  to  $v$  and if they are in different subsets there is no such path. The subgraphs  $G_1 = (V_1, E_1, \phi_1), \dots, G_m = (V_m, E_m, \phi_m)$  induced by the sets  $V_1, \dots, V_m$  are called the *connected components* of  $G$ . Every edge of  $G$  appears in one of the connected components. To see this, suppose that  $\{u, v\}$  is an edge and note that the edge is a path from  $u$  to  $v$  and so  $u$  and  $v$  are in the same induced subgraph,  $G_i$ . By the definition of induced subgraph,  $\{u, v\}$  is in  $G_i$ .

**Example 11 (Connected components as an equivalence relation)** You may have noticed that the “definition” that we have given of connected components is a bit sloppy: We need to know that the partitioning into such subsets can actually occur. To see that this is not trivially obvious, define two integers to be “connected” if they have a common factor. Thus 2 and 6 are connected and 3 and 6 are connected, but 2 and 3 are not connected and so we cannot partition the set  $V = \{2, 3, 6\}$  into “connected components”. We must use some property of the definition of graphs and paths to show that the partitioning of vertices is possible. One way to do this is to construct an equivalence relation.

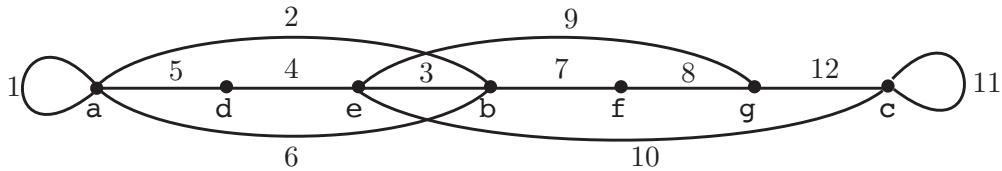
For  $u, v \in V$ , write  $u \sim v$  if and only if either  $u = v$  or there is a walk from  $u$  to  $v$ . It is clear that  $\sim$  is reflexive and symmetric. We now prove that it is transitive. Let  $u \sim v \sim w$ . The walk from  $u$  to  $v$  followed by the walk from  $v$  to  $w$  is a walk from  $u$  to  $w$ . This completes the proof that  $u \sim v$  is an equivalence relation. The relation partitions  $V$  into subsets  $V_1, \dots, V_m$ . By Theorem 2, the vertex induced subgraphs of the  $V_i$  satisfy Definition 10.  $\square$

When talking about connectivity, graphs and digraphs are different. In a digraph, the fact that there is a directed walk from  $u$  to  $v$  does not, in general, imply that there is a

## Basic Concepts in Graph Theory

directed walk from  $v$  to  $u$ . Thus, the “directed walk relation”, unlike the “walk relation” is not symmetric. This complicates the theory of connectivity for digraphs.

**Example 12 (Eulerian graphs)** We are going to describe a process for constructing a graph  $G = (V, E, \phi)$  (with loops allowed). Start with  $V = \{v_1\}$  consisting of a single vertex and with  $E = \emptyset$ . Add an edge  $e_1$ , with  $\phi(e_1) = \{v_1, v_2\}$ , to  $E$ . If  $v_1 = v_2$ , we have a graph with one vertex and one edge (a loop), else we have a graph with two vertices and one edge. Keep track of the vertices and edges in the order added. Here  $(v_1, v_2)$  is the sequence of vertices in the order added and  $(e_1)$  is the sequence of edges in order added. Suppose we continue this process to construct a sequence of vertices (not necessarily distinct) and sequence of *distinct* edges. At the point where  $k$  distinct edges have been added, if  $v$  is the last vertex added, then we add a new edge  $e_{k+1}$ , different from all previous edges, with  $\phi(e_{k+1}) = \{v, v'\}$  where either  $v'$  is a vertex already added or a new vertex. Here is a picture of this process carried out with the edges numbered in the order added



where the vertex sequence is

$$S = (a, a, b, e, d, a, b, f, g, e, c, c, g).$$

Such a graph is called a graph with an *Eulerian trail*. The edges, in the order added, are the Eulerian trail and  $S$  is the vertex sequence of the trail

By construction, if  $G$  is a graph with an *Eulerian trail*, then there is a trail in  $G$  that includes every edge in  $G$ . If there is a circuit in  $G$  that includes every edge of  $G$  then  $G$  is called an *Eulerian circuit graph* or graph with an *Eulerian circuit*. Thinking about the above example, if a graph has an Eulerian trail but no Eulerian circuit, then all vertices of the graph have even degree except the start vertex ( $a$  in our example with degree 5) and end vertex ( $g$  in our example with degree 3). If a graph has an Eulerian circuit then all vertices have even degree. The converses in each case are also true (but take a little work to show): If  $G$  is a connected graph in which every vertex has even degree then  $G$  has an Eulerian circuit. If  $G$  is a connected graph with all vertices but two of even degree, then  $G$  has an Eulerian trail joining the two vertices of odd degree.  $\square$

Here is a precise definition of Eulerian trail and circuit.

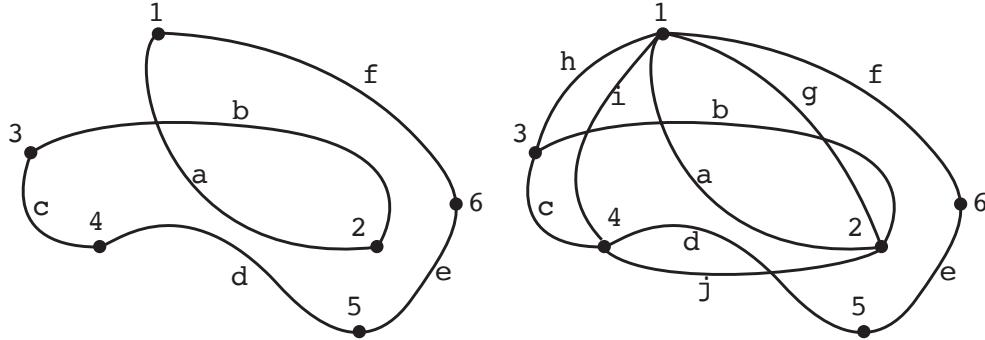
**Definition 11 (Eulerian trail, circuit)** Let  $G = (V, E, \phi)$  be a connected graph. If there is a trail with edge sequence  $(e_1, e_2, \dots, e_k)$  in  $G$  which uses each edge in  $E$ , then  $(e_1, e_2, \dots, e_k)$  is called an *Eulerian trail*. If there is a circuit  $C = (V', E', \phi')$  in  $G$  with  $E' = E$ , then  $C$  is called an *Eulerian circuit*.

The ideas of a directed Eulerian circuit and directed Eulerian trail for directed graphs are defined in exactly the same manner.

## Section 2: Digraphs, Paths, and Subgraphs

An Eulerian circuit in a graph contains every edge of that graph. What about a cycle that contains every vertex but not necessarily every edge? Our next example discusses that issue.

**Example 13 (Hamiltonian cycle)** Start with a graph  $G' = (V, E', \phi')$  that is a cycle and then add additional edges, without adding any new vertices, to obtain a graph  $G = (V, E, \phi)$ . As an example, consider



where the first graph  $G' = (V, E', \phi')$  is the cycle induced by the edges  $\{a, b, c, d, e, f\}$ . The second graph  $G = (V, E, \phi)$  is obtained from  $G'$  by adding edges  $g, h, i$  and  $j$ . A graph that can be constructed from such a two-step process is called a *Hamiltonian graph*. The cycle  $G'$  is called a *Hamiltonian cycle for*  $G$ .

**Definition 12 (Hamiltonian cycle, Hamiltonian graph)** A cycle in a graph  $G = (V, E, \phi)$  is a *Hamiltonian cycle for*  $G$  if every element of  $V$  is a vertex of the cycle. A graph  $G = (V, E, \phi)$  is *Hamiltonian* if it has a subgraph that is a Hamiltonian cycle for  $G$ .

Notice that an Eulerian circuit uses every edge exactly once and a Hamiltonian cycle uses every vertex exactly once. We gave a very simple characterization of when a graph has an Eulerian circuit (in terms of degrees of vertices). There is no simple characterization of when a graph has a Hamiltonian cycle. On the contrary, the issue of whether or not a graph has a Hamiltonian cycle is notoriously difficult to resolve in general.

As we already mentioned, connectivity issues in digraphs are much more difficult than in graphs. A digraph is *strongly connected* if, for every two vertices  $v$  and  $w$  there is a directed path from  $v$  to  $w$ . From any digraph  $D$ , we can construct a simple graph  $S(D)$  on the same set of vertices by letting  $\{v, w\}$  be an edge of  $S(D)$  if and only if at least one of  $(u, v)$  and  $(v, u)$  is an edge of  $D$ . You should be able to show that if  $D$  is strongly connected then  $S(D)$  is connected. The converse is false. As an example, take  $D = (V, E)$  to be the simple digraph where  $V = \{1, 2\}$  and  $E = \{(1, 2)\}$ . There is no directed path from 2 to 1, but clearly  $S(D) = (V, \{\{1, 2\}\})$  is connected.

Other issues for digraphs analogous to those for graphs work out pretty well, but are more technical. An example is the notion of degree for vertices. For any subset  $U$  of the vertices  $V$  of a directed graph  $D = (V, E)$ , define  $d_{\text{in}}(U)$  to be the number of edges of  $e$  of  $D$  with  $\phi(e)$  of the form  $(w, u)$  where  $u \in U$  and  $w \notin U$ . Define  $d_{\text{out}}(U)$  similarly. If  $U = \{v\}$  consists of just one vertex,  $d_{\text{in}}(U)$  is usually written simply as  $d_{\text{in}}(v)$  rather than

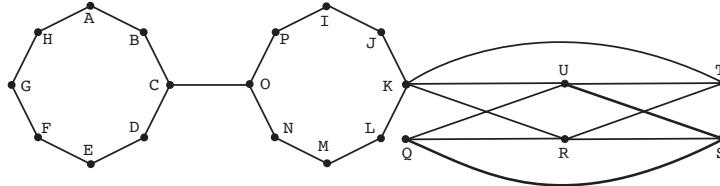
## Basic Concepts in Graph Theory

the more technically correct  $d_{\text{in}}(\{v\})$ . Similarly, we write  $d_{\text{out}}(v)$ . You should compute  $d_{\text{in}}(v)$  and  $d_{\text{out}}(v)$  for the vertices  $v$  of the graph of Example 9. You should be able to show that  $\sum d_{\text{in}}(v) = \sum d_{\text{out}}(v) = |E|$ , where the sums range over all  $v \in V$ . See the Exercises for Section 1 for the idea.

**Example 14 (Biconnectivity of graphs)** Let  $G = (V, E, \phi)$  be a graph. For  $e, f \in E$  write  $e \sim f$  if either  $e = f$  or there is a cycle of  $G$  that contains both  $e$  and  $f$ . We claim that this is an equivalence relation. The reflexive and symmetric parts are easy. Suppose that  $e \sim f \sim g$ . If  $e = g$ , then  $e \sim g$ , so suppose that  $e \neq g$ . Let  $\phi(e) = \{v_1, v_2\}$ . Let  $C(e, f)$  be the cycle containing  $e$  and  $f$  and  $C(f, g)$  the cycle containing  $f$  and  $g$ . In  $C(e, f)$  there is a path  $P_1$  from  $v_1$  to  $v_2$  that does not contain  $e$ . Let  $x$  and  $y \neq x$  be the first and last vertices on  $P_1$  that lie on the cycle containing  $f$  and  $g$ . We know that there must be such points because the edge  $f$  is on  $P_1$ . Let  $P_2$  be the path in  $C(e, f)$  from  $y$  to  $x$  containing  $e$ . In  $C(f, g)$  there is a path  $P_3$  from  $x$  to  $y$  containing  $g$ . We claim that  $P_2$  followed by  $P_3$  defines a cycle containing  $e$  and  $g$ .

Some examples may help. Consider a graph that consists of two disjoint cycles that are joined by an edge. There are three biconnected components — each cycle and the edge joining them. Now consider three cycles that are disjoint except for one vertex that belongs to all three of them. Again there are three biconnected components — each of the cycles.

Since  $\sim$  is an equivalence relation on the edges of  $G$ , it partitions them. If the partition has only one block, then we say that  $G$  is a *biconnected graph*. If  $E'$  is a block in the partition, the subgraph of  $G$  induced by  $E'$  is called a *biconnected component* of  $G$ . Note that the biconnected components of  $G$  are not necessarily disjoint: Biconnected components may have vertices in common (but *never* edges). There are four biconnected components in the following graph. Two are the cycles, one is the edge  $\{C, O\}$ , and the fourth consists of all of the rest of the edges.

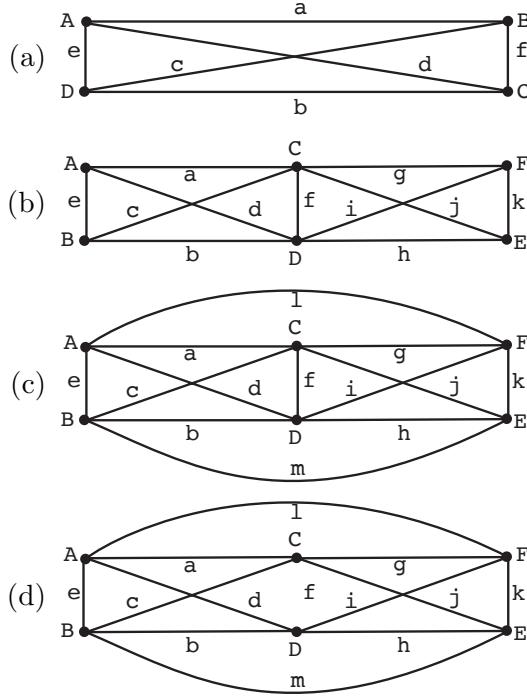


## Exercises for Section 2

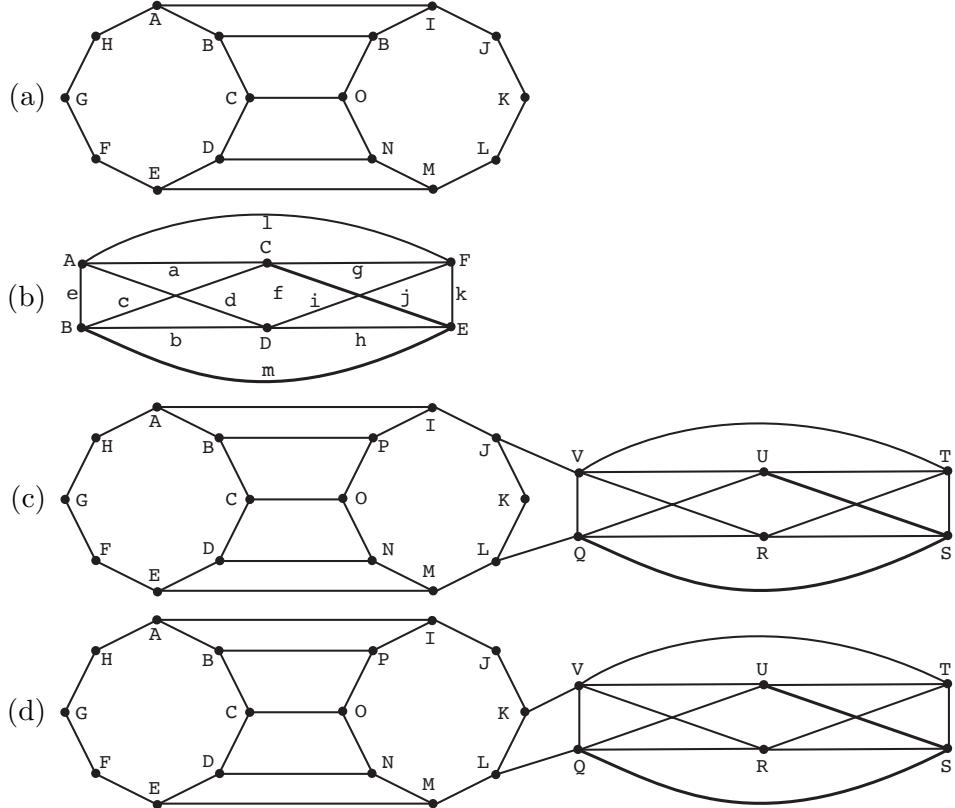
- 2.1.** A graph  $G = (V, E)$  is called *bipartite* if  $V$  can be partitioned into two sets  $C$  and  $S$  such that each edge has one vertex in  $C$  and one vertex in  $S$ . As a specific example, let  $C$  be the set of courses at the university and  $S$  the set of students. Let  $V = C \cup S$  and let  $\{s, c\} \in E$  if and only if student  $s$  is enrolled in course  $c$ .
- Prove that  $G = (V, E)$  is a simple graph.
  - Prove that every cycle of  $G$  has an even number of edges.

- 2.2.** In each of the following graphs, find the longest trail (most edges) and longest circuit. If the graph has an Eulerian circuit or trail, say so.

## Section 2: Digraphs, Paths, and Subgraphs



- 2.3.** For each of the following graphs  $G = (V, E, \phi)$ , find a cycle in  $G$  of maximum length. State whether or not the graph is Hamiltonian.



- 2.4.** We are interested in the number of simple digraphs with  $V = \underline{n}$

## Basic Concepts in Graph Theory

- (a) Find the number of them.
  - (b) Find the number of them with no loops.
  - (c) In both cases, find the number of them with exactly  $q$  edges.
- 2.5.** An *oriented simple graph* is a simple graph which has been converted to a digraph by assigning an orientation to each edge. The orientation of  $\{u, v\}$  can be thought of as a mapping of it to either  $(u, v)$  or  $(v, u)$ .
- (a) Give an example of a simple digraph that has no loops but is not an oriented simple graph
  - (b) Find the number of oriented simple digraphs.
  - (c) Find the number of them with exactly  $q$  edges.
- 2.6.** A binary relation  $R$  on  $S$  is an *order relation* if it is reflexive, antisymmetric, and transitive.  $R$  is *antisymmetric* if for all  $(x, y) \in R$  with  $x \neq y$ ,  $(y, x) \notin R$ . Given an order relation  $R$ , the covering relation  $H$  of  $R$  consists of all  $(x, z) \in R$ ,  $x \neq z$ , such that there is no  $y$ , distinct from both  $x$  and  $z$ , such that  $(x, y) \in R$  and  $(y, z) \in R$ . A pictorial representation of the covering relation as a directed graph is called a “Hasse diagram” of  $H$ .
- (a) Show that the divides relation on

$$S = \{2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16\}$$

is an order relation. By definition,  $(x, y)$  is in the divides relation on  $S$  if  $x$  is a factor of  $y$ . Thus,  $(4, 12)$  is in the divides relation.  $x|y$  is the standard notation for  $x$  is a factor of  $y$ .

- (b) Find and draw a picture of the directed graph of the covering relation of the divides relation.  
*Hint:* You must find all pairs  $(x, z) \in S \times S$  such that  $x|z$  but there does not exist any  $y$ ,  $x < y < z$ , such that  $x|y$  and  $y|z$ .

---

## Section 3: Trees

Trees play an important role in a variety of algorithms. We have used decision trees to enhance our understanding of recursion. In this section, we define trees precisely and look at some of their properties.

**Definition 13 (Tree)** If  $G$  is a connected graph without any cycles then  $G$  is called a tree. (If  $|V| = 1$ , then  $G$  is connected and hence is a tree.) A tree is also called a free tree.

The graph of Example 2 is connected but is not a tree. It has many cycles, including  $(\{A, B, C\}, \{a, e, c\})$ . The subgraph of this graph induced by the edges  $\{a, e, g\}$  is a tree. If

### Section 3: Trees

$G$  is a tree, then  $\phi$  is an injection since if  $e_1 \neq e_2$  and  $\phi(e_1) = \phi(e_2)$ , then  $\{e_1, e_2\}$  induces a cycle. In other words, any graph with parallel edges is not a tree. Likewise, a loop is a cycle, so a tree has no loops. Thus, we can think of a tree as a simple graph when we are not interested in names of the edges.

Since the notion of a tree is so important, it will be useful to have some equivalent definitions of a tree. We state them as a theorem

**Theorem 4 (Alternative definitions of a tree)**    *If  $G$  is a connected graph, the following are equivalent.*

- (a)  $G$  is a tree.
- (b)  $G$  has no cycles.
- (c) For every pair of vertices  $u \neq v$  in  $G$ , there is exactly one path from  $u$  to  $v$ .
- (d) Removing any edge from  $G$  gives a graph which is not connected.
- (e) The number of vertices of  $G$  is one more than the number of edges of  $G$ .

**Proof:** We are given that  $G$  is connected, thus, by the definition of a tree, (a) and (b) are equivalent.

Theorem 3 can be used to prove that (b) implies (c). We leave that as an exercise (show **not** (c) implies **not** (b)).

If  $\{u, v\}$  is an edge, it follows from (c) that the edge is the only path from  $u$  to  $v$  and so removing it disconnects the graph. Hence (c) implies (d).

We leave it as an exercise to prove that (d) implies (b) (show **not** (b) implies **not** (d)).

Thus far, we have shown (a) and (b) are equivalent, and we have shown that (b) implies (c) implies (d) implies (b), so (a), (b), (c), and (d) are all equivalent. All that remains is to include (e) in this equivalence class of statements. Do do this, all we have to do is show that (e) implies any of the equivalent statements (a), (b), (c), and (d) and, conversely, some one of (a), (b), (c), and (d) implies (e). We shall show that (b) implies (e) and that (e) implies (a).

We first show that (b) implies (e). We will use induction on the number of vertices of  $G$ . If  $G$  has one vertex, it has no edges and (e) is satisfied. Otherwise, we claim that  $G$  has a vertex  $u$  of degree 1; that is, it lies on only one edge  $\{u, w\}$ . We prove this claim shortly. Remove  $u$  and  $\{u, v\}$  to obtain a graph  $H$  with one less edge and one less vertex. Since  $G$  is connected and has no cycles, the same is true of  $H$ . By the induction hypothesis,  $H$  has one less edge than vertex. Since we got from  $G$  to  $H$  by removing one vertex and one edge,  $G$  must also have one less edge than vertex. By induction, the proof is done. It remains to prove the existence of  $u$ . Suppose no such  $u$  exists; that is, suppose that each vertex lies on at least two edges. We will derive a contradiction. Start at any vertex  $v_1$  of  $G$  leave  $v_1$  by some edge  $e_1$  to reach another vertex  $v_2$ . Leave  $v_2$  by some edge  $e_2$  different from the edge used to reach  $v_2$ . Continue with this process. Since each vertex lies on at least two edges, the process never stops. Hence we eventually repeat a vertex, say

$$v_1, e_1, v_2, \dots, v_k, e_k, \dots, v_n, e_n, v_{n+1} = v_k.$$

## Basic Concepts in Graph Theory

The edges  $e_k, \dots, e_n$  form a cycle, which is a contradiction.

Having shown that (b) implies (e), we now show that (e) implies (a). We use the contrapositive and show that **not** (a) implies **not** (e). Thus we assume  $G$  is not a tree. Hence, by (d) we can remove an edge from  $G$  to get a new graph which is still connected. If this is not a tree, repeat the process and keep doing so until we reach a tree  $T$ . For a tree  $T$ , we trivially satisfy (a) which implies (b) and (b) implies (e). Thus, the number of vertices is now one more than the number of edges in the graph  $T$ . Since, in going from  $G$  to  $T$ , we removed edges from  $G$  but did not remove vertices,  $G$  must have at least as many edges as vertices. This shows **not** (a) implies **not** (e) and completes the proof.  $\square$

**Definition 14 (Forest)** A forest is a graph all of whose connected components are trees. In particular, a forest with one component is a tree. (Connected components were defined following Definition 10.)

**Example 15 (A relation for forests)** Suppose a forest has  $v$  vertices,  $e$  edges and  $c$  (connected) components. What values are possible for the triple of numbers  $(v, e, c)$ ? It might seem at first that almost anything is possible, but this is not so. In fact  $v - c = e$  because of Theorem 4(e). Why? Let the forest consist of trees  $T_1, \dots, T_c$  and let the triples for  $T_i$  be  $(v_i, e_i, c_i)$ . Since a tree is connected,  $c_i = 1$ . By the theorem,  $e_i = v_i - 1$ . Since  $v = v_1 + \dots + v_c$  and  $e = e_1 + \dots + e_c$  we have

$$e = (v_1 - 1) + (v_2 - 1) + \dots + (v_c - 1) = (v_1 + \dots + v_c) - c = v - c.$$

Suppose a forest has  $e = 12$  and  $v = 15$ . We know immediately that it must be made up of three trees because  $c = v - e = 15 - 12$ .

Suppose we know that a graph  $G = (V, E, \phi)$  has  $v = 15$  and  $c = 3$ , what is the fewest edges it could have? For each component of  $G$ , we can remove edges one by one until we cannot remove any more without breaking the component into two components. At this point, we are left with each component a tree. Thus we are left with a forest of  $c = 3$  trees that still has  $v = 15$  vertices. By our relation  $v - c = e$ , this forest has 12 edges. Since we may have removed edges from the original graph to get to this forest, the original graph has at least 12 edges.

What is the maximum number of edges that a graph  $G = (V, E, \phi)$  with  $v = 15$  and  $c = 3$  could have? Since we allow multiple edges, a graph could have an arbitrarily large number of edges for a fixed  $v$  and  $c$  — if  $e$  is an edge with  $\phi(e) = \{u, v\}$ , add in as many edges  $e_i$  with  $\phi(e_i) = \{u, v\}$  as you wish. Hence we will have to insist that  $G$  be a simple graph.

What is the maximum number of edges that a simple graph  $G$  with  $v = 15$  and  $c = 3$  could have? This is a bit trickier. Let's start with a graph where  $c$  is not specified. The edges in a simple graph are a subset of  $\mathcal{P}_2(V)$  and since  $\mathcal{P}_2(V)$  has  $\binom{v}{2}$  elements, a simple graph with  $v$  vertices has at most  $\binom{v}{2}$  edges.

Now let's return to the case when we know there must be three components in our simple graph. Suppose the number of vertices in the components are  $v_1, v_2$  and  $v_3$ . Since there are no edges between components, we can look at each component by itself. Using

### Section 3: Trees

the result in the previous paragraph for each component, the maximum number of possible edges is  $\binom{v_1}{2} + \binom{v_2}{2} + \binom{v_3}{2}$ . We don't know  $v_1, v_2, v_3$ . All we know is that they are strictly positive integers that sum to  $v$ . It turns out that the maximum occurs when one of  $v_i$  is as large as possible and the others equal 1, but the proof is beyond this course. Thus the answer is  $\binom{v-2}{2}$ , which in our case is  $\binom{13}{2} = 78$ . In general, if there were  $c$  components,  $c-1$  components would have one vertex each and the remaining component would have  $v - (c-1) = v + 1 - c$  vertices. Hence there can be no more than  $\binom{v+1-c}{2}$  edges.

Reviewing what we've done, we see:

- There is no graph  $G = (V, E, \phi)$  with  $v - c > e$ .
- If  $v - c = e$ , the graph is a forest of  $c$  trees and any such forest will do as an example.
- If  $v - c < e$ , there are many examples, none of which are forests.
- If  $v - c < e$  and we have a *simple* graph, then we must have  $e \leq \binom{v+1-c}{2}$ .  $\square$

Recall that decision trees, as we have used them, have some special properties. First, they have a starting point. Second, the edges (decisions) out of each vertex are ordered. We now formalize these concepts.

**Definition 15 (Rooted graph)** A pair  $(G, v)$ , consisting of a graph  $G = (V, E, \phi)$  and a specified vertex  $v$ , is called a *rooted graph* with root  $v$ .

**Definition 16 (Parent, child, sibling and leaf)** Let  $(T, r)$  be a rooted tree. If  $w$  is any vertex other than  $r$ , let  $r = v_0, v_1, \dots, v_k, v_{k+1} = w$ , be the list of vertices on the unique path from  $r$  to  $w$ . We call  $v_k$  the *parent* of  $w$  and call  $w$  a *child* of  $v_k$ . Parents and children are also called *fathers* and *sons*. Vertices with the same parent are *siblings*. A vertex with no children is a *leaf*. All other vertices are *internal vertices* of the tree.

**Definition 17 (Rooted plane tree)** Let  $(T, r)$  be a rooted tree. For each vertex, order the children of the vertex. The result is a *rooted plane tree*, which we abbreviate to *RP-tree*. RP-trees are also called *ordered trees*. An RP-tree is also called, in certain contexts, a *decision tree*, and, when there is no chance of misunderstanding, simply a *tree*.

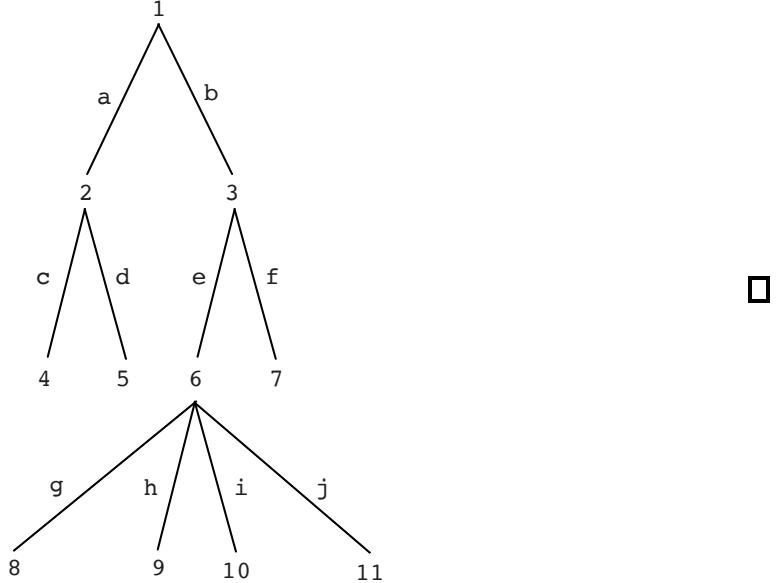
Since almost all trees in computer science are rooted and plane, computer scientists usually call a rooted plane tree simply a tree. It's important to know what people mean!

**Example 16 (A rooted plane tree)** Below is a picture of a rooted plane tree  $T = (V, E, \phi)$ . In this case  $V = \underline{11}$  and  $E = \{a, \dots, j\}$ . There are no parallel edges or loops, as required by the definition of a RP-tree. The root is  $r = 1$ . For each vertex, there is a unique path from the root to that vertex. Since  $\phi$  is an injection, once  $\phi$  has been defined (as it is in the picture), that unique path can be specified by the vertex sequence alone. Thus, the path from the root to 6 is  $(1, 3, 6)$ . The path from the root to 9 is  $(1, 3, 6, 9)$ . Sometimes computer scientists refer to the path from the root to a vertex  $v$  as the "stack" of  $v$ .

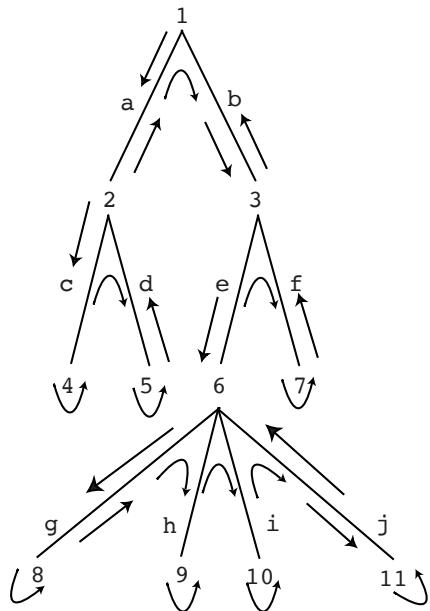
## Basic Concepts in Graph Theory

In the tree below, the vertex 6 is the parent of the vertex 9. The vertices 8, 9, 10, and 11 are the children of 6 and, they are siblings of each other. The leaves of the tree are 4, 5, 7, 8, 9, 10, and 11. All other vertices (including the root) are internal vertices of the tree.

Remember, an RP-tree is a tree with added properties. Therefore, it must satisfy (a) through (e) of Theorem 4. In particular,  $T$  has no cycles. Also, there is a unique path between any two vertices (e.g., the path from 5 to 8 is  $(5, 2, 1, 3, 6, 8)$ ). Removing any edge gives a graph which is not connected (e.g., removing  $j$  disconnects  $T$  into a tree with 10 vertices and a tree with 1 vertex; removing  $e$  disconnects  $T$  into a tree with 6 vertices and one with 5 vertices). Finally, the number of edges (10) is one less than the number of vertices.



**Example 17 (Traversing a rooted plane tree)** Just as in the case of decision trees, one can define the notion of *depth first* traversals of a RP-tree.



### Section 3: Trees

Imagine going around (“traversing”) the above RP-tree following arrows. Start at the root, 1, go down edge  $a$  to vertex 2, etc. Here is the sequence of vertices as encountered in this process: 1, 2, 4, 2, 5, 2, 1, 3, 6, 8, 6, 9, 6, 10, 6, 11, 6, 3, 7, 3, 1. This sequence of vertices is called the *depth first vertex sequence*,  $\text{DFV}(T)$ , of the RP-tree  $T$ . The number of times each vertex appears in  $\text{DFV}(T)$  is one plus the number of children of that vertex. For edges, the corresponding sequence is  $a, c, c, d, d, a, b, e, g, g, h, h, i, i, j, j, e, f, f, b$ . This sequence is the *depth first edge sequence*,  $\text{DFE}(T)$ , of the tree. Every edge appears exactly twice in  $\text{DFE}(T)$ . If the vertices of the RP-tree are read left to right, top to bottom, we obtain the sequence 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11. This is called the *breadth first vertex sequence*,  $\text{BFV}(T)$ . Similarly, the *breadth first edge sequence*,  $\text{BFE}(T)$ , is  $a, b, c, d, e, f, g, h, i, j$ .

The sequences  $\text{BFV}(T)$  and  $\text{BFE}(T)$  are linear orderings of the vertices and edges of the RP-tree  $T$  (i.e., each vertex or edge appears exactly once in the sequence). We also associate linear orderings with  $\text{DFV}(T)$  called the *preorder sequence of vertices* of  $T$ ,  $\text{PREV}(T)$ , and the *postorder sequence of vertices* of  $T$ ,  $\text{POSV}(T)$ .

$\text{PREV}(T) = 1, 2, 4, 5, 3, 6, 8, 9, 10, 11, 7$  is the sequence of *first* occurrences of the vertices of  $T$  in  $\text{DFV}(T)$ .

$\text{POSV}(T) = 4, 5, 2, 8, 9, 10, 11, 6, 7, 3, 1$  is the sequence of *last* occurrences of the vertices of  $T$  in  $\text{DFV}(T)$ .

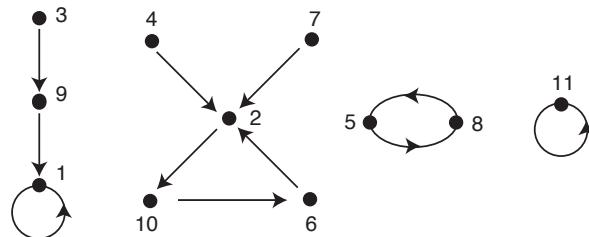
Notice that the order in which the leaves of  $T$  appear, 4, 5, 8, 9, 10, 11, is the same in both  $\text{PREV}(T)$  and  $\text{POSV}(T)$ . Can you see why this is always true for any tree?  $\square$

**\*Example 18 (The number of labeled trees)** How many  $n$ -vertex labeled trees are there? In other words, count the number of trees with vertex set  $V = \underline{n}$ . The answer has been obtained in a variety of ways. We will do it by establishing a correspondence between trees and functions by using digraphs.

Suppose  $f$  is a function from  $V$  to  $V$ . We can represent this as a simple digraph  $(V, E)$  where the edges are  $\{(v, f(v)) \mid v \in V\}$ . The function

$$\begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 \\ 1 & 10 & 9 & 2 & 8 & 2 & 2 & 5 & 1 & 6 & 11 \end{pmatrix}$$

corresponds to the directed graph

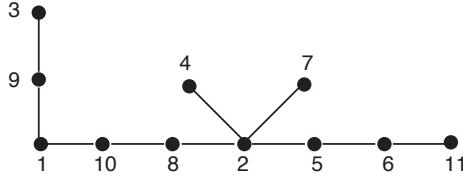


Such graphs are called *functional digraphs*. You should be able to convince yourself that a functional digraph consists of cycles (including loops) with each vertex on a cycle being the root of a tree of noncyclic edges. The edges of the trees are directed toward the roots. In the previous figure,

## Basic Concepts in Graph Theory

- 1 is the root of the tree with vertex set  $\{1, 3, 9\}$ ,
- 2 is the root of the tree with vertex set  $\{2, 4, 7\}$ ,
- 5 is the root of the tree with vertex set  $\{5\}$ ,
- 6 is the root of the tree with vertex set  $\{6\}$ ,
- 8 is the root of the tree with vertex set  $\{8\}$ ,
- 10 is the root of the tree with vertex set  $\{10\}$  and
- 11 is the root of the tree with vertex set  $\{11\}$ .

In a tree, there is a unique path from the vertex 1 to the vertex  $n$ . Remove all the edges on the path and list the vertices on the path, excluding 1 and  $n$ , in the order they are encountered. Interpret this list as a permutation in 1 line form. Draw the functional digraph for the cycle form, adding the cycles  $(1)$  and  $(n)$ . Add the trees that are attached to each of the cycle vertices, directing their edges toward the cycle vertices. Consider the following figure.



The one line form is  $10, 8, 2, 5, 6$ . In two line form it is  $\begin{pmatrix} 2 & 5 & 6 & 8 & 10 \\ 10 & 8 & 2 & 5 & 6 \end{pmatrix}$ . Thus the cycle form is  $(2, 10, 6)(5, 8)$ . When we add the two cycles  $(1)$  and  $(11)$  to this, draw the directed graph, and attach the directed trees, we obtain the functional digraph pictured earlier.

We leave it to you to convince yourself that this gives us a one-to-one correspondence between trees with  $V = \underline{n}$  and functions  $f : \underline{n} \rightarrow \underline{n}$  with  $f(1) = 1$  and  $f(n) = n$ . In creating such a function, there are  $n$  choices for each of  $f(2), \dots, f(n - 1)$ . Thus there are  $n^{n-2}$  such functions and hence  $n^{n-2}$  trees.  $\square$

---

## Spanning Trees

Trees are not only important objects of study per se, but are important as special subgraphs of general graphs. A *spanning tree* is one such subgraph. For notational simplicity, we shall restrict ourselves to simple graphs,  $G = (V, E)$ , in the following discussion. The ideas we discuss extend easily to graphs  $G = (V, E, \phi)$ , even allowing loops.

**Definition 18 (Spanning tree)** A *spanning tree* of a (simple) graph  $G = (V, E)$  is a subgraph  $T = (V, E')$  which is a tree and has the same set of vertices as  $G$ .

**Example 19 (Connected graphs and spanning trees)** Since a tree is connected, a graph with a spanning tree must be connected. On the other hand, it is not hard to see that every connected graph has a spanning tree. Any simple graph  $G = (V, E)$  has a subgraph that is a tree,  $T' = (V', E')$ . Take  $V' = \{v\}$  to be one vertex and  $E'$  empty. Suppose that  $T' = (V', E')$  is the largest such “subtree.” If  $T'$  is not a spanning tree then there is a vertex  $w$  of  $G$  that is not a vertex of  $T'$ . If  $G$  is connected, choose a vertex  $u$  in  $T'$  and a path  $w = x_1, x_2, \dots, e_k = u$  from  $w$  to  $u$ . Let  $j$ ,  $1 < j \leq k$ , be the first integer such that  $x_j$  is a vertex of  $T'$ . Then adding the edge  $\{x_{j-1}, x_j\}$  and the vertex  $x_{j-1}$  to  $T'$  creates a subtree  $T$  of  $G$  that is larger than  $T'$ , a contradiction of the maximality of  $T'$ . We have, in fact, shown that a graph is connected if and only if every maximal subtree is a spanning tree. Thus we have: A graph is connected if and only if it has a spanning tree. It follows that, if we had an algorithm that was guaranteed to find a spanning tree whenever such a tree exists, then this algorithm could be used to decide if a graph is connected.  $\square$

**Example 20 (Minimum spanning trees)** Suppose we wish to install “lines” to link various sites together. A site may be a computer installation, a town, or a factory. A line may be a digital communication channel, a rail line or, a shipping route for supplies. We’ll assume that

- (a) a line operates in both directions;
- (b) it must be possible to get from any site to any other site using lines;
- (c) each possible line has a cost (rental rate, construction cost, or shipping cost) independent of each other line’s cost;
- (d) we want to choose lines to minimize the total cost.

We can think of the sites as vertices  $V$  in a (simple) graph, the possible lines as edges  $E$  and the costs as a function  $\lambda$  from the edges to the positive real numbers. Because of (a) and (b), the lines  $E' \subseteq E$  we actually choose will be such that  $T = (V, E')$  is connected. Because of (d),  $T$  will be a spanning tree since, if it had more edges, we could delete some, but if we delete any from a tree it will not be connected by Theorem 4.  $\square$

We now formalize these ideas in a definition:

**Definition 19 (Weights in a graph)** Let  $G = (V, E)$  be a simple graph and let  $\lambda$  be a function from  $E$  to the positive real numbers. We call  $\lambda(e)$  the weight of the edge  $e$ . If  $H = (V', E')$  is a subgraph of  $G$ , then  $\lambda(H)$ , the weight of  $H$ , is the sum of  $\lambda(e')$  over all  $e' \in E'$ .

A minimum weight spanning tree for a connected graph  $G$  is a spanning tree such that  $\lambda(T) \leq \lambda(T')$  whenever  $T'$  is another spanning tree.

How can we find a minimum weight spanning tree  $T$ ? One approach is to construct  $T$  by adding an edge at a time in a greedy way. Since we want to minimize the weight, “greedy” means keeping the weight of each edge we add as low as possible. Here’s such an algorithm.

**Theorem 5 (Minimum weight spanning tree: Prim’s algorithm)** Let  $G = (V, E)$  be a simple graph with edge weights given by  $\lambda$ . If the algorithm stops with  $V' \neq V$ ,  $G$  has no spanning tree; otherwise,  $(V, E')$  is a minimum weight spanning tree for  $G$ .

## Basic Concepts in Graph Theory

1. **Start:** Let  $E' = \emptyset$  and let  $V' = \{v_0\}$  where  $v_0$  is any vertex in  $V$ .
2. **Possible Edges:** Let  $F \subseteq E$  be those edges  $f = \{x, y\}$  with one vertex in  $V'$  and one vertex not in  $V'$ . If  $F = \emptyset$ , stop.
3. **Choose Edge Greedily:** Let  $f = \{x, y\}$  be such that  $\lambda(f)$  is a minimum over all  $f \in F$ . Replace  $V'$  with  $V' \cup \{x, y\}$  and  $E'$  with  $E' \cup \{f\}$ . Go to Step 2.

**Proof:** We begin with the first part; i.e., if the algorithm stops with  $V' \neq V$ , then  $G$  has no spanning tree. The argument is similar to that used in Example 19. Suppose that  $V' \neq V$  and that there is a spanning tree. We will prove that the algorithm does not stop at  $V'$ . Choose  $u \in V - V'$  and  $v \in V'$ . Since  $G$  is connected, there must be a path from  $u$  to  $v$ . Each vertex on the path is either in  $V'$  or not. Since  $u \notin V'$  and  $v \in V'$ , there must be an edge  $f$  on the path with one end in  $V'$  and one end not in  $V'$ . But then  $f \in F$  and so the algorithm does not stop at  $V'$ .

We now prove that, if  $G$  has a spanning tree, then  $(V, E')$  is a minimum weight spanning tree. One way to do this is by induction: We will prove that at each step there is a minimum weight spanning tree of  $G$  that contains  $E'$ .

The starting case for the induction is the first step in the algorithm; i.e.,  $E' = \emptyset$ . Since  $G$  has a spanning tree, it must have a minimum weight spanning tree. The edges of this tree obviously contain the empty set, which is what  $E'$  equals at the start.

We now carry out the inductive step of the proof. Let  $V'$  and  $E'$  be the values going into Step 3 and let  $f = \{x, y\}$  be the edge chosen there. By the induction hypothesis, there is a minimum weight spanning tree  $T$  of  $G$  that contains the edges  $E'$ . If it also contains the edge  $f$ , we are done. Suppose it does not contain  $f$ . We will prove that we can replace an edge in the minimum weight tree with  $f$  and still achieve minimum weight.

Since  $T$  contains all the vertices of  $G$ , it contains  $x$  and  $y$  and, also, some path  $P$  from  $x$  to  $y$ . Suppose  $x \in V'$  and  $y \notin V'$ , this path must contain an edge  $e = \{u, v\}$  with  $u \in V'$  and  $v \notin V'$ . We now prove that removing  $e$  from  $T$  and then adding  $f$  to  $T$  will still give a minimum spanning tree.

By the definition of  $F$  in Step 2,  $e \in F$  and so, by the definition of  $f$ ,  $\lambda(e) \geq \lambda(f)$ . Thus the weight of the tree does not increase. If we show that the result is still a tree, this will complete the proof.

The path  $P$  together with the edge  $f$  forms a cycle in  $G$ . Removing  $e$  from  $P$  and adding  $f$  still allows us to reach every vertex in  $P$  and so the altered tree is still connected. It is also still a tree because it contains no cycles — adding  $f$  created only one cycle and removing  $e$  destroyed it. This completes the proof that the algorithm is correct.  $\square$

The algorithm for finding a minimum weight spanning tree that we have just proved is sometimes referred to as *Prim's Algorithm*. A variation on this algorithm, proved in a similar manner, is called *Kruskal's algorithm*. In Kruskal's algorithm, step 2 of Prim's algorithm is changed to

- 2'. **Possible Edges:** Let  $F \subseteq E$  be those edges  $f = \{x, y\}$  where  $x$  and  $y$  do not belong to the same component of  $(V, E')$ . If  $F = \emptyset$ , stop.

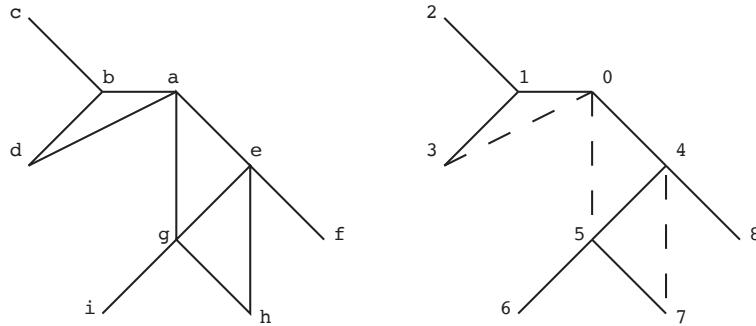
Intuitively,  $f \notin F$  if  $f$  forms a cycle with any collection of edges from  $E'$ . Otherwise,  $f \in F$ . This extra freedom is sometimes convenient. Our next example gives much less freedom in

choosing new edges to add to the spanning tree, but produces a type of spanning tree that is useful in many algorithms applicable to computer science.

**Example 21 (Algorithm for lineal or depth-first spanning trees)** We start with a rooted simple graph  $G = (V, E)$  with  $v_0$  as root. The algorithmic process constructs a spanning tree rooted at  $v_0$ . It follows the same general form as Theorem 5. The weights, if there, are ignored.

1. **Start:** Let  $E' = \emptyset$  and let  $V' = \{v_0\}$  where  $v_0$  is the root of  $G$ . Let  $T' = (V', E')$  be the starting subtree, rooted at  $v_0$ .
2. **Possible New Edge:** Let  $v$  be the last vertex added to  $V'$  where  $T' = (V', E')$  is the subtree thus far constructed, with root  $v_0$ . Let  $x$  be the first vertex on the unique path from  $v$  to  $v_0$  for which there is an edge  $f = \{x, y\}$  with  $x \in V'$  and  $y \notin V'$ . If there is no such  $x$ , stop.
3. **Add Edge :** Replace  $V'$  with  $V' \cup \{y\}$  and  $E'$  with  $E' \cup \{f\}$  to obtain  $T' = (V', E')$  as the new subtree thus far constructed, with root  $v_0$ . (Note:  $y$  is now the last vertex added to  $V'$ .) Go to Step 2.

Here is an example. We are going to find a lineal spanning tree for the graph below, root  $a$ . The result is shown on the right where the original vertices have been replaced by the order in which they have been added to the “tree thus far constructed” in the algorithm.



When there is a choice, we choose the left or upward vertex. For example, at the start, when  $b, d, e$  and  $g$  are all allowed, we choose  $b$ . When vertex 2 was added, the path to the root was  $(2, 1, 0)$ . We went along this path towards the root and found that at 1, a new edge to 3 could be added. Now the path to the root became  $(3, 1, 0)$  and we had to go all of the way to 0 to add a new edge (the edge to 4). You should go through the rest of the algorithm. Although there are some choices, the basic rule of step 2 of the algorithm must always be followed.

There are two extremely important properties that this algorithm has

1. When the rooted spanning tree  $T$  for  $G$  has been constructed, there may be edges of  $G$  not in the spanning tree. In the above picture, there are three such edges, indicated by dashed lines. If  $\{x, y\}$  is such an edge, then either  $x$  lies on the path from  $y$  to the root or the other way around. For example, the edge  $\{4, 7\}$  in the example has 4 on the path from 7 to the root 0. This is the “lineal” property from which the spanning trees of this class get their name.

## Basic Concepts in Graph Theory

2. If, when the rooted spanning tree  $T$  has been constructed, the vertices of  $T$  are labeled in the order added by the algorithm **AND** the children of each vertex of  $T$  are ordered by the same numbering, then an RP-tree is the result. For this RP tree, the numbers on the vertices correspond to preorder, PREV( $T$ ), of vertices on this tree (starting with the root having value 0). Check this out for the above example.

We will not prove that the algorithm we have presented has properties 1 and 2. We leave it to you to study the example, construct other examples, and come to an intuitive understanding of these properties.  $\square$

Property 1 in the preceding example is the basis for the formal definition of a lineal spanning tree:

**Definition 20 (Lineal or depth-first spanning tree)** *Let  $x$  and  $y$  be two vertices in a rooted tree with root  $r$ . If  $x$  is on the path connecting  $r$  to  $y$ , we say that  $y$  is a descendant of  $x$ . (In particular, all vertices are descendants of  $r$ .) If one of  $u$  and  $v$  is a descendant of the other, we say that  $\{u, v\}$  is a lineal pair. A lineal spanning tree or depth-first spanning tree of a connected graph  $G = (V, E)$  is a rooted spanning tree of  $G$  such that each edge  $\{u, v\}$  of  $G$  is a lineal pair.*

In our example, vertices  $\{6, 7\}$  are not a lineal pair relative to the rooted tree constructed. But  $\{4, 7\}$ , which is an edge of  $G$ , is a lineal pair. Trivially, the vertices of any edge of the tree  $T$  form a lineal pair.

We close this section by proving a theorem using lineal spanning trees. We don't "overexplain" this theorem to encourage you to think about the properties of lineal spanning trees that make the proof much simpler than what we might have come up with without lineal spanning trees. Recall that a graph  $G = (V, E)$  is called *bipartite* if  $V$  can be partitioned into two sets  $C$  and  $S$  such that each edge has one vertex in  $C$  and one vertex in  $S$  (Exercises for Section 2).

**Theorem 6 (Bipartite and cycle lengths)** *Let  $G = (V, E)$  be a simple graph.  $G$  is bipartite if and only if every cycle has even length.*

**Proof:** If  $G$  has a cycle of odd length, label each vertex with the block of some proposed bipartite partition  $\{C, S\}$ . For example, if  $(x_1, x_2, x_3)$  are the vertices, in some order, of a cycle of length three, then the block labels (start with  $C$ ) would be  $(C, S, C)$ . This would mean that the edge  $\{x_1, x_3\}$  would have both vertices in block  $C$ . This violates the definition of a bipartite graph. Since this problem happens for any cycle of odd length, a bipartite graph can never contain a cycle of odd length.

To prove the converse, we must show that if every cycle of  $G$  has even length, then  $G$  is bipartite. Suppose every cycle of  $G$  has even length. Choose a vertex  $v_0$  as root of  $G$  and construct a lineal spanning tree  $T$  for  $G$  with root  $v_0$ . Label the root  $v_0$  of  $T$  with  $C$ , all vertices of  $T$  of distance 1 from  $v_0$  with  $S$ , all of distance 2 from  $v_0$  with  $C$ , etc. Put vertices labeled  $C$  into block  $C$  of a partition  $\{C, S\}$  of  $V$ , put all other vertices into block  $S$ . If  $f = \{x, y\}$  is an edge of  $T$  then  $x$  and  $y$  are in different blocks of the partition  $\{C, S\}$ .

by construction. If  $f = \{x, y\}$  is an edge of  $G$  not in  $T$  then the two facts (1)  $T$  is lineal and (2) every cycle has even length, imply that  $x$  and  $y$  are in different blocks of the partition  $\{C, S\}$ . This completes the proof.  $\square$

---

### Exercises for Section 3

**3.1.** In this exercise, we study how counting edges and vertices in a graph can establish that cycles exist. For parts (a) and (b), let  $G = (V, E, \phi)$  be a graph with loops allowed.

- (a) Using induction on  $n$ , prove:

If  $n \geq 0$ ,  $G$  is connected and  $G$  has  $v$  vertices and  $v + n$  edges, then  $G$  has at least  $n + 1$  cycles.

- (b) Prove that, if  $G$  has  $v$  vertices,  $e$  edges and  $c$  components, then  $G$  has at least  $c + e - v$  cycles.

*Hint:* Use (a) for each component.

- (c) Show that (a) is best possible, even for simple graphs. In other words, for each  $n$  construct a simple graph that has  $n$  more edges than vertices but has only  $n + 1$  cycles.

**3.2.** Let  $T = (V, E)$  be a tree and let  $d(v)$  be the degree of a vertex

- (a) Prove that  $\sum_{v \in V} (2 - d(v)) = 2$ .

- (b) Prove that, if  $T$  has a vertex of degree  $m \geq 2$ , then it has at least  $m$  vertices of degree 1.

- (c) Give an example for all  $m \geq 2$  of a tree with a vertex of degree  $m$  and only  $m$  leaves.

**3.3.** Give an example of a graph that satisfies the specified condition or show that no such graph exists.

- (a) A tree with six vertices and six edges

- (b) A tree with three or more vertices, two vertices of degree one and all the other vertices with degree three or more.

- (c) A disconnected graph with 10 vertices and 8 edges.

- (d) A disconnected graph with 12 vertices and 11 edges and no cycle.

- (e) A tree with 6 vertices and the sum of the degrees of all vertices 12.

- (f) A connected graph with 6 edges, 4 vertices, and exactly 2 cycles.

- (g) A graph with 6 vertices, 6 edges and no cycles.

**3.4.** The *height* of a rooted tree is the maximum height of any leaf. The length of the unique path from a leaf of the tree to the root is, by definition, the height of that

## Basic Concepts in Graph Theory

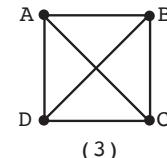
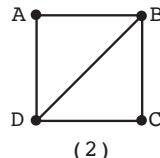
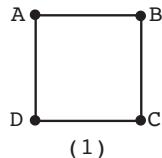
leaf. A rooted tree in which each non-leaf vertex has at most two children is called a *binary tree*. If each non-leaf vertex has exactly two children, the tree is called a *full binary tree*.

- (a) If a binary tree has  $l$  leaves and height  $h$  prove that  $l \leq 2^h$ . (Taking logarithms gives  $\log_2(l) \leq h$ .)
- (b) A binary tree has  $l$  leaves. What can you say about the maximum value of  $h$ ?
- (c) Given a full binary tree with  $l$  leaves, what is the maximum height  $h$ ?
- (d) Given a full binary tree with  $l$  leaves, what is the minimum height  $h$ ?
- (e) Given a binary tree of  $l$  leaves, what is the minimum height  $h$ ?

**3.5.** In each of the following cases, state whether or not such a tree is possible.

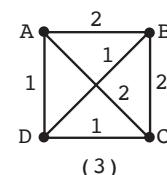
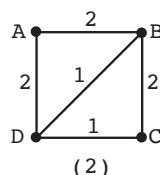
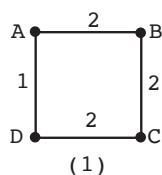
- (a) A binary tree with 35 leaves and height 100.
- (b) A full binary tree with 21 leaves and height 21.
- (c) A binary tree with 33 leaves and height 5.
- (d) A rooted tree of height 5 where every internal vertex has 3 children and there are 365 vertices.

**3.6.** For each of the following graphs:



- (a) Find all spanning trees.
- (b) Find all spanning trees up to isomorphism.
- (c) Find all depth-first spanning trees rooted at  $A$ .
- (d) Find all depth-first spanning trees rooted at  $B$ .

**3.7.** For each of the following graphs:

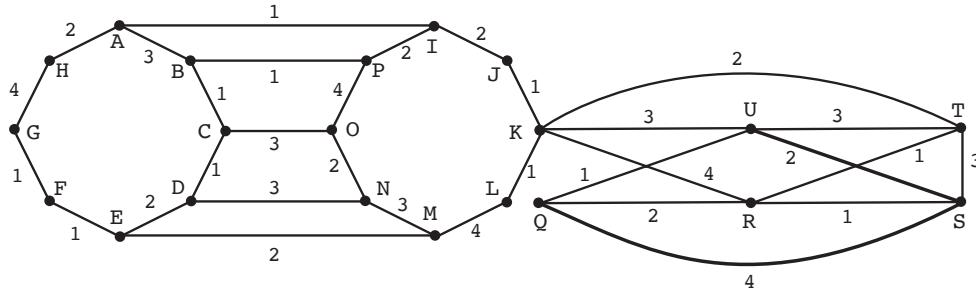


- (a) Find all minimum spanning trees.
- (b) Find all minimum spanning trees up to isomorphism.

## Section 4: Rates of Growth and Analysis of Algorithms

- (c) Among all depth-first spanning trees rooted at  $A$ , find those of minimum weight.
- (d) Among all depth-first spanning trees rooted at  $B$ , find those of minimum weight.

**3.8.** In the following graph, the edges are weighted either 1, 2, 3, or 4.



Referring to Theorem 5 and the discussion following of Kruskal's algorithm:

- (a) Find a minimum spanning tree using Prim's algorithm
  - (b) Find a minimum spanning tree using Kruskal's algorithm.
  - (c) Find a depth-first spanning tree rooted at  $K$ .
- 

## Section 4: Rates of Growth and Analysis of Algorithms

Suppose we have an algorithm and someone asks us “How good is it?” To answer that question, we need to know what they mean. They might mean “Is it correct?” or “Is it understandable?” or “Is it easy to program?” We won’t deal with any of these.

They also might mean “How fast is it?” or “How much space does it need?” These two questions can be studied by similar methods, so we’ll just focus on speed. Even now, the question is not precise enough. Does the person mean “How fast is it on this particular problem and this particular machine using this particular code and this particular compiler?” We could answer this simply by running the program! Unfortunately, that doesn’t tell us what would happen with other machines or with other problems that the algorithm is designed to handle.

We would like to answer a question such as “How fast is Algorithm 1 for finding a spanning tree?” in such a way that we can compare that answer to “How fast is Algorithm 2 for finding a spanning tree?” and obtain something that is not machine or problem dependent. At first, this may sound like an impossible goal. To some extent it is; however, quite a bit can be said.

How do we achieve machine independence? We think in terms of simple machine operations such as multiplication, fetching from memory and so on. If one algorithm uses fewer of these than another, it should be faster. Those of you familiar with computer

## Basic Concepts in Graph Theory

instruction timing will object that different basic machine operations take different amounts of time. That's true, but the times are not wildly different. Thus, if one algorithm uses *a lot fewer* operations than another, it should be faster. It should be clear from this that we can be a bit sloppy about what we call an operation; for example, we might call something like  $x = a + b$  one operation. On the other hand, we can't be so sloppy that we call  $x = a_1 + \dots + a_n$  one operation if  $n$  is something that can be arbitrarily large.

**Example 22 (Finding the maximum)** Let's look at how long it takes to find the maximum of a list of  $n$  integers where we know nothing about the order they are in or how big the integers are. Let  $a_1, \dots, a_n$  be the list of integers. Here's our algorithm for finding the maximum.

```
max = a1
For i = 2, ..., n
    If ai > max, then max = ai.
End for
Return max
```

Being sloppy, we could say that the entire comparison and replacement in the “If” takes an operation and so does the stepping of the index  $i$ . Since this is done  $n - 1$  times, we get  $2n - 2$  operations. There are some setup and return operations, say  $s$ , giving a total of  $2n - 2 + s$  operations. Since all this is rather sloppy all we can really say is that for large  $n$  and actual code on an actual machine, the procedure will take about  $Cn$  “ticks” of the machine’s clock. Since we can’t determine  $C$  by our methods, it will be helpful to have a notation that ignores it. We use  $\Theta(f(n))$  to designate any function that behaves like a constant times  $f(n)$  for arbitrarily large  $n$ . Thus we would say that the “If” takes time  $\Theta(n)$  and the setup and return takes time  $\Theta(1)$ . Thus the total time is  $\Theta(n) + \Theta(1)$ . Since  $n$  is much bigger than 1 for large  $n$ , the total time is  $\Theta(n)$ .  $\square$

We need to define  $\Theta$  more precisely and list its most important properties. We will also find it useful to define  $O$ , read “big oh.”

**Definition 21 (Notation for  $\Theta$  and  $O$ )** Let  $f, g$  and  $h$  be functions from the positive integers to the nonnegative real numbers. We say that  $g(n)$  is  $\Theta(f(n))$  if there exist positive constants  $A$  and  $B$  such that  $Af(n) \leq g(n) \leq Bf(n)$  for all sufficiently large  $n$ . In this case we say that  $f$  and  $g$  grow at the same rate. We say that  $h(n)$  is  $O(f(n))$  if there exists a positive constant  $B$  such that  $h(n) \leq Bf(n)$  for all sufficiently large  $n$ . In this case we say that  $h$  grows no faster than  $f$  or, equivalently, that  $f$  grows at least as fast as  $h$ .

The phrase “ $S(n)$  is true for all sufficiently large  $n$ ” means that there is some integer  $N$  such that  $S(n)$  is true whenever  $n \geq N$ . Saying that something is  $\Theta(f(n))$  gives an idea of *how big it is* for large values of  $n$ . Saying that something is  $O(f(n))$  gives an idea of *an upper bound on how big it is* for all large values of  $n$ . (We said “idea of” because we don’t know what the constants  $A$  and  $B$  are.)

**Theorem 7 (Some properties of  $\Theta$  and  $O$ )** We have

- If  $g(n)$  is  $\Theta(f(n))$ , then  $g(n)$  is  $O(f(n))$ .

## Section 4: Rates of Growth and Analysis of Algorithms

- (b)  $f(n)$  is  $\Theta(f(n))$  and  $f(n)$  is  $O(f(n))$ .
- (c) If  $g(n)$  is  $\Theta(f(n))$  and  $C$  and  $D$  are positive constants, then  $Cg(n)$  is  $\Theta(Df(n))$ .  
If  $g(n)$  is  $O(f(n))$  and  $C$  and  $D$  are positive constants, then  $Cg(n)$  is  $O(Df(n))$ .
- (d) If  $g(n)$  is  $\Theta(f(n))$ , then  $f(n)$  is  $\Theta(g(n))$ .
- (e) If  $g(n)$  is  $\Theta(f(n))$  and  $f(n)$  is  $\Theta(h(n))$ , then  $g(n)$  is  $\Theta(h(n))$ .  
If  $g(n)$  is  $O(f(n))$  and  $f(n)$  is  $O(h(n))$ , then  $g(n)$  is  $O(h(n))$ .
- (f) If  $g_1(n)$  is  $\Theta(f_1(n))$ ,  $g_2(n)$  is  $\Theta(f_2(n))$ , then  $g_1(n)+g_2(n)$  is  $\Theta(\max(f_1(n), f_2(n)))$ .  
If  $g_1(n)$  is  $O(f_1(n))$ ,  $g_2(n)$  is  $O(f_2(n))$ , then  $g_1(n)+g_2(n)$  is  $O(\max(f_1(n), f_2(n)))$ .

Note that as a consequence of properties (b), (d) and (e) above, the statement “ $g(n)$  is  $\Theta(f(n))$ ” defines an equivalence relation on the set of functions from the positive integers to the nonnegative reals. As with any equivalence relation, we can think of it globally as partition into equivalence classes or locally as a relation between pairs of elements in the set on which the equivalence relation is defined. In the former sense “ $g(n)$  is  $\Theta(f(n))$ ” means that “ $g(n)$  belongs to the equivalence class  $\Theta(f(n))$  associated with  $f$ .” In the latter sense, “ $g(n)$  is  $\Theta(f(n))$ ” means  $g \sim_{\Theta} f$  where  $\sim_{\Theta}$  is an equivalence relation called “is  $\Theta$ .”

**Proof:** Most of the proofs are left as an exercise. We'll do (e) for  $\Theta$ . We are given that there are constants  $A_i$  and  $B_i$  such that

$$A_1 f(n) \leq g(n) \leq B_1 f(n)$$

and

$$A_2 h(n) \leq f(n) \leq B_2 h(n)$$

for all sufficiently large  $n$ . It follows that

$$A_1 A_2 h(n) \leq A_1 f(n) \leq g(n) \leq B_1 f(n) \leq B_1 B_2 h(n)$$

for all sufficiently large  $n$ . With  $A = A_1 A_2$  and  $B = B_1 B_2$ , it follows that  $g(n)$  is  $\Theta(h(n))$ .  $\square$

**Example 23 (Additional observations on  $\Theta$  and  $O$ )** In this example, we have collected some additional information about our notation.

**Functions which are not always positive.** Our definitions of  $\Theta$  and  $O$  are only for functions whose values are nonnegative. The definitions can be extended to arbitrary functions by using absolute values; e.g.,  $A|f(n)| \leq |g(n)| \leq B|f(n)|$  means  $g(n) = \Theta(f(n))$ . All the results in the theorem still hold except (f) for  $\Theta$ . This observation is most often applied to the case where the function  $f$  is “eventually” nonnegative ( $\exists M$  such that  $\forall n > M, f(n) \geq 0$ ). This is the case, for example with any polynomial in  $n$  with positive coefficient for the highest power of  $n$ .

**Taking limits.** When comparing two well-behaved functions  $f(n)$  and  $g(n)$ , limits can be helpful:

$$\lim_{n \rightarrow \infty} \frac{g(n)}{f(n)} = C > 0 \quad \text{implies} \quad g(n) \text{ is } \Theta(f(n))$$

## Basic Concepts in Graph Theory

and

$$\lim_{n \rightarrow \infty} \frac{g(n)}{f(n)} = C \geq 0 \quad \text{implies} \quad g(n) \quad \text{is} \quad O(f(n)).$$

We assume here that the function  $f$  is never zero past some integer  $N$  so that the ratio is defined. The constants  $A$  and  $B$  of the definition can, in the first case, be taken to be  $C - \epsilon$  and  $C + \epsilon$ , where  $\epsilon$  is any positive number ( $\epsilon = 1$  is a simple choice). In the second case, take  $B$  to be  $C + \epsilon$ . If, in the first case,  $C = 1$ , then  $f$  and  $g$  are said to be *asymptotic* or asymptotically equal. This is written  $f \sim g$ . If, in the second case,  $C = 0$ , then  $g$  is said to be *little oh* of  $f$  (written  $g = o(f)$ ). We will not use the “asymptotic” and “little oh” concepts.

**Polynomials.** In particular, you can take any polynomial as  $f(n)$ , say  $f(n) = a_k n^k + \dots + a_0$ , and any other polynomial as  $g(n)$ , say  $g(n) = b_k n^k + \dots + b_0$ . For  $f$  and  $g$  to be eventually positive we must have both  $a_k$  and  $b_k$  positive. If that is so, then  $g(n)$  is  $\Theta(f(n))$ . Note in particular that we must have  $g(n)$  is  $\Theta(n^k)$ .

**Logarithms.** Two questions that arise concerning logarithms are (a) “What base should I use?” and (b) “How fast do they grow?”

The base does not matter because  $\log_a x = (\log_a b)(\log_b x)$  and constant factors like  $\log_a b$  are ignored in  $\Theta()$  and  $O()$ .

It is known from calculus that  $\log n \rightarrow \infty$  as  $n \rightarrow \infty$  and that  $\lim_{n \rightarrow \infty} (\log n)/n^\epsilon = 0$  for every  $\epsilon > 0$ . Thus logarithms grow, but they grow slower than powers of  $n$ . For example,  $n \log n$  is  $O(n^{3/2})$  but  $n^{3/2}$  is not  $O(n \log n)$ .

**A proof.** How do we prove

$$\lim_{n \rightarrow \infty} \frac{g(n)}{f(n)} = C > 0 \quad \text{implies} \quad g(n) \text{ is } \Theta(f(n))?$$

By definition, the limit statement means that for any  $\epsilon > 0$  there exists  $N$  such that for all  $n > N$ ,  $|\frac{g(n)}{f(n)} - C| < \epsilon$ . If  $\epsilon \geq C$ , replace it with a smaller  $\epsilon$ . From  $|\frac{g(n)}{f(n)} - C| < \epsilon$ , for all  $n > N$ ,

$$C - \epsilon < \frac{g(n)}{f(n)} < C + \epsilon, \quad \text{or} \quad (C - \epsilon)f(n) < \frac{g(n)}{f(n)} < (C + \epsilon)f(n).$$

Take  $A = (C - \epsilon)$  and  $B = (C + \epsilon)$  in the definition of  $\Theta$ .  $\square$

**Example 24 (Using  $\Theta$ )** To illustrate these ideas, we’ll consider three algorithms for evaluating a polynomial  $p(x)$  of degree  $n$  at some point  $r$ ; i.e., computing  $p_0 + p_1 r + \dots + p_n r^n$ . We are interested in how fast they are when  $n$  is large. Here are the procedures. You should convince yourself that they work.

```

Poly1( $n, p, r$ )
   $S = p_0$ 
  For  $i = 1, \dots, n$        $S = S + p_i * \text{Pow}(r, i)$ .
  Return  $S$ 
End

```

## Section 4: Rates of Growth and Analysis of Algorithms

```

Pow( $r, i$ )
   $P = 1$ 
  For  $j = 1, \dots, n$      $P = P * r.$ 
  Return  $P$ 
End

Poly2( $n, p, r$ )
   $S = p_0$ 
   $P = 1$ 
  For  $i = 1, \dots, n$ 
     $P = P * r.$ 
     $S = S + p_i * P$ 
  End for
  Return  $S$ 
End

Poly3( $n, p, r$ )
   $S = p_n$ 
  For  $i = n, \dots, 2, 1$      $S = S * r + p_{i-1}$ 
  Return  $S$ 
End

```

Let  $T_n(\text{Name})$  be the time required for the procedure Name. Let's analyze Poly1. The “**For**” loop in Pow is executed  $i$  times and so takes  $Ci$  operations for some constant  $C$ . The setup and return in Pow takes some constant number of operations  $D$ . Thus  $T_n(\text{Pow}) = Ci + D$  operations. As a result, the  $i$ th iteration of the “**For**” loop in Poly1 takes  $Ci + E$  operations for some constants  $C$  and  $E > D$ . Adding this over  $i = 1, 2, \dots, n$ , we see that the total time spent in the “**For**” loop is  $\Theta(n^2)$  since  $\sum_{i=1}^n i = n(n+1)/2$ . (This requires using some of the properties of  $\Theta$ . You should write out the details.) Since the rest of Poly1 takes  $\Theta(1)$  time,  $T_n(\text{Poly1})$  is  $\Theta(n^2)$ .

The amount of time spent in the “For” loop of Poly2 is constant and the loop is executed  $n$  times. It follows that  $T_n(\text{Poly2})$  is  $\Theta(n)$ . The same analysis applies to Poly3.

What can we conclude from this about the comparative speed of the algorithms? By the definition of  $\Theta$ , there are positive reals  $A$  and  $B$  so that  $An^2 \leq T_n(\text{Poly1})$  and  $T_n(\text{Poly2}) \leq Bn$  for sufficiently large  $n$ . Thus  $T_n(\text{Poly2})/T_n(\text{Poly1}) \leq B/An$ . As  $n$  gets larger, Poly2 looks better and better compared to Poly1.

Unfortunately, the crudeness of  $\Theta$  does not allow us to make any distinction between Poly2 and Poly3. What we can say is that  $T_n(\text{Poly2})$  is  $\Theta(T_n(\text{Poly3}))$ ; i.e.,  $T_n(\text{Poly2})$  and  $T_n(\text{Poly3})$  grow at the same rate. A more refined estimate can be obtained by counting the actual number of operations involved.  $\square$

So far we have talked about how long an algorithm takes to run as if this were a simple, clear concept. In the next example we'll see that there's an important point that we've ignored.

**\*Example 25 (What is average running time?)** Let's consider the problem of (a) deciding whether or not a simple graph can be properly colored with four colors and, (b) if a proper coloring exists, producing one. A *proper coloring* of a simple graph

## Basic Concepts in Graph Theory

$G = (V, E)$  is a function  $\lambda: V \rightarrow C$ , the set of “colors,” such that, if  $\{u, v\}$  is an edge, then  $\lambda(u) \neq \lambda(v)$ . We may as well assume that  $V = \underline{n}$  and that the colors are  $c_1, c_2, c_3$  and  $c_4$ .

Here’s a simple algorithm to determine a  $\lambda$  by using backtracking to go lexicographically through possible colorings  $\lambda(1), \lambda(2), \dots, \lambda(n)$ .

1. **Initialize:** Set  $v = 1$  and  $\lambda(1) = c_1$ .
2. **Advance in decision tree:** If  $v = n$ , stop with  $\lambda$  determined; otherwise, set  $v = v + 1$  and  $\lambda(v) = c_1$ .
3. **Test:** If  $\lambda(i) \neq \lambda(v)$  for all  $i < v$  for which  $\{i, v\} \in E$ , go to Step 2.
4. **Select next decision:** Let  $j$  be such that  $\lambda(v) = c_j$ . If  $j < 4$ , set  $\lambda(v) = c_{j+1}$  and go to Step 3.
5. **Backtrack:** If  $v = 1$ , stop with coloring impossible; otherwise, set  $v = v - 1$  and go to Step 4.

How fast is this algorithm? Obviously it will depend on the graph. Here are two extreme cases:

- Suppose the subgraph induced by the first five vertices is the complete graph  $K_5$  (i.e., all of the ten possible edges are present). The algorithm stops after trying to color the first five vertices and discovering that there is no proper coloring. Thus the running time does not depend on  $n$  and so is in  $\Theta(1)$ .
- Suppose that the first  $n - 5$  vertices have no edges and that the last five vertices induce  $K_5$ . The algorithm tries all possible assignments of colors to the first  $n - 5$  vertices and, for each of them, discovers that it cannot properly color the last five because they form  $K_5$ . Thus the algorithm makes between  $4^{n-5}$  and  $4^n$  assignments of colors and so its running time is  $\Theta(4^n)$  — a much faster growing time than  $\Theta(1)$ .

What should we do about studying the running time of such an algorithm? It’s reasonable to talk about the *average* time the algorithm takes if we expect to give it lots of graphs to look at. Most  $n$  vertex graphs will have many sets of five vertices that induce  $K_5$ . (We won’t prove this.) As a result, the algorithm has running time in  $\Theta(1)$  for most graphs. In fact, it can be proved that the average number of assignments of the form  $\lambda(v) = c_k$  that are made is  $\Theta(1)$  and so the average running time is  $\Theta(1)$ . This means that the average running time of the algorithm is bounded for all  $n$ , which is quite good!

Now suppose you give this algorithm to a friend, telling her that the average running time is bounded. She thanks you profusely for such a wonderful algorithm and puts it to work coloring randomly generated “planar” graphs. These are a special class of graphs whose pictures can be drawn in the plane without edges crossing each other. (All trees are planar, but  $K_5$  is not planar.) By a famous theorem called the *Four Color Theorem*, every planar graph can be properly colored with four colors, so the algorithm will find the coloring. To do so it must make assignments of the form  $\lambda(v) = c_k$  for each vertex  $v$ . Thus it must make at least  $n$  assignments. (Actually it will almost surely make *many, many* more.) Your friend soon comes back to you complaining that your algorithm takes a long time to run. What went wrong?

You were averaging over all simple graphs with  $n$  vertices. Your friend was averaging over all simple planar graphs with  $n$  vertices. The average running times are *very* different! There is a lesson here:

You must be VERY clear what you are averaging over.

## Section 4: Rates of Growth and Analysis of Algorithms

Because situations like this do occur in real life, computer scientists are careful to specify what kind of running time they are talking about; either the average of the running time over some reasonable, clearly specified set of problems or the worst (longest) running time over all possibilities.  $\square$

You should be able to see that saying something is  $\Theta(\cdot)$  leaves a lot out because we have no idea of the constants that are omitted. How can we compare two algorithms? Here are two rules of thumb.

- If one algorithm is  $\Theta(f(n))$  and the other is  $\Theta(g(n))$ , the algorithm with the slower growing function ( $f$  or  $g$ ) is probably the better choice.
- If both algorithms are  $\Theta(f(n))$ , the algorithm with the simpler data structures is probably better.

These rules are far from foolproof, but they provide some guidance.

---

### \*Polynomial Time Algorithms

Computer scientists talk about “polynomial time algorithms.” What does this mean? Suppose that the algorithm can handle arbitrarily large problems and that it takes  $\Theta(n)$  seconds on a problem of “size”  $n$ . Then we call it a linear time algorithm. More generally, if there is a (possibly quite large) integer  $k$  such that the worst case running time on a problem of “size”  $n$  is  $O(n^k)$ , then we say the algorithm is polynomial time.

You may have noticed the quotes around size and wondered why. It is necessary to specify what we mean by the size of a problem. Size is often interpreted as the number of bits required to specify the problem in binary form. You may object that this is imprecise since a problem can be specified in many ways. This is true; however, the number of bits in one “reasonable” representation doesn’t differ too much from the number of bits in another. We won’t pursue this further.

If the worst case time for an algorithm is polynomial, theoretical computer scientists think of this as a good algorithm. (This is because polynomials grow relatively slowly; for example, exponential functions grow much faster.) The problem that the algorithm solves is called *tractable*.

Do there exist *intractable problems*; i.e., problems for which no polynomial time algorithm can ever be found? Yes, but we won’t study them here. More interesting is the fact that there are a large number of practical problems for which

- no polynomial time algorithm is known and
- no one has been able prove that the problems are intractable.

We’ll discuss this a bit.

Consider the following problems.

- **Coloring problem:** For any  $c > 2$ , devise an algorithm whose input can be any simple graph and whose output answers the question “Can the graph be properly colored in  $c$  colors?”

## Basic Concepts in Graph Theory

- **Traveling salesman problem:** For any  $B$ , devise an algorithm whose input can be any  $n > 0$  and any real valued edge labeling,  $\lambda: \mathcal{P}_2(\underline{n}) \rightarrow \mathbb{R}$ , for  $K_n$ , the complete graph on  $n$  vertices. The algorithm must answer the question “Is there a cycle through all  $n$  vertices with cost  $B$  or less?” (The cost of a cycle is the sum of  $\lambda(e)$  over all  $e$  in the cycle.)
- **Clique problem:** Given a simple graph  $G = (V, E)$  and an integer  $s$ , is there a subset  $S \subseteq V$ ,  $|S| = s$ , whose induced subgraph is the complete graph on  $S$  (i.e., a subgraph of  $G$  with vertex set  $S$  and with  $\binom{s}{2}$  edges)?

No one knows if these problems are tractable, but it is known that, if one is tractable, then they all are. There are hundreds more problems that people are interested in which belong to this particular list in which all or none are tractable. These problems are called *NP-complete problems*. Many people regard deciding if the NP-complete problems are tractable to be the foremost open problem in theoretical computer science.

The NP-complete problems have an interesting property which we now discuss. If the algorithm says “yes,” then there must be a specific example that shows why this is so (an assignment of colors, a cycle, an automaton). There is no requirement that the algorithm actually produce such an example. Suppose we somehow obtain a coloring, a cycle or an automaton which is claimed to be such an example. Part of the definition of NP-complete requires that we be able to check the claim in polynomial time. Thus we can check a purported example quickly but, so far as is known, it may take a long time to determine if such an example exists. In other words, I can check your guesses quickly but I don’t know how to tell you quickly if any examples exist.

There are problems like the NP-complete problems where no one knows how to do any checking in polynomial time. For example, modify the traveling salesman problem to ask for the minimum cost cycle. No one knows how to verify in polynomial time that a given cycle is actually the minimum cost cycle. If the modified traveling salesman problem is tractable, so is the one we presented above: You need only find the minimum cost cycle and compare its cost to  $B$ . Such problems are called *NP-hard* because they are at least as hard as NP-complete problems. A problem which is tractable if the NP-complete problems are tractable is called *NP-easy*.

Some problems are both NP-easy and NP-hard but may not be NP-complete. Why is this? NP-complete problems must ask a “yes/no” type of question and it must be possible to check a specific example in polynomial time as noted in the previous paragraph. We discuss an example.

\***Example 26 (Chromatic number)** The *chromatic number*  $\chi(G)$  of a graph  $G$  is the least number of colors needed to properly color  $G$ . The problem of deciding whether a graph can be properly colored with  $c$  colors is NP-complete. The problem of determining  $\chi(G)$  is NP-hard. If we know  $\chi(G)$ , then we can determine if  $c$  colors are enough by checking if  $c \geq \chi(G)$ .

The problem of determining  $\chi(G)$  is also NP-easy. You can color  $G$  with  $c$  colors if and only if  $c \geq \chi(G)$ . We know that  $0 \leq \chi(G) \leq n$  for a graph with  $n$  vertices. Ask if  $c$  colors suffice for  $c = 0, 1, 2, \dots$ . The least  $c$  for which the answer is “yes” is  $\chi(G)$ . Thus the worst case time for finding  $\chi(G)$  is at most  $n$  times the worst case time for the NP-complete problem. Hence one time is O of a polynomial in  $n$  if and only if the other is.  $\square$

## Section 4: Rates of Growth and Analysis of Algorithms

What can we do if we cannot find a good algorithm for a problem? There are three main types of partial algorithms:

1. **Almost good:** It is polynomial time for all but a very small subset of possible problems. (If we are interested in all graphs, our coloring algorithm in Example 25 is almost good for any fixed  $c$ .)
2. **Almost correct:** It is polynomial time but in some rare cases does not find the correct answer. (If we are interested in all graphs and a fixed  $c$ , automatically reporting that a large graph can't be colored with  $c$  colors is almost correct — but it is rather useless.) In some situations, a fast almost correct algorithm can be useful.
3. **Close:** It is a polynomial time algorithm for a minimization problem and comes close to the true minimum. (There are useful close algorithms for approximating the minimum cycle in the Traveling Salesman Problem.)

Some of the algorithms make use of random number generators in interesting ways. Unfortunately, further discussion of these problems is beyond the scope of this text.

---

### \*A Theorem for Recursive Algorithms

Some algorithms, such as merge sorting, call themselves. This is known as a *recursive algorithm* or a *divide and conquer algorithm*.

When we try estimate the running time of such algorithms, we obtain a recursion. In Section 2 of Unit DT, we examined the problem of solving recursions. We saw that finding exact solutions to recursions is difficult. The recursions that we obtain for algorithms are not covered by the methods in that section. Furthermore, the recursions are often not known exactly because we may only be able to obtain an estimate of the form  $\Theta(\ )$  for some of the work. The next example illustrates this problem.

**\*Example 27 (Sorting by recursive merging)** Given a list  $L$  of  $n$  items, we wish to sort it. Here is the merge sorting algorithm from Section 3 of Unit DT.

```
Sort(L)
  If length is 1, return L
  Else
    Split L into two lists L1 and L2
    S1 = Sort(L1)
    S2 = Sort(L2)
    S = Merge(L1, L2)
    Return S
  End if
End
```

We need to be more specific about how the lists are split. Let  $m$  be  $n/2$  rounded down, let  $L1$  be the first  $m$  items in  $L$  and let  $L2$  be the last  $n - m$  items in  $L$ .

## Basic Concepts in Graph Theory

One way to measure the running time of Sort(L) is to count the number of comparisons that are required. Let this number be  $T(n)$ . We would like to know how fast  $T(n)$  grows as a function of  $n$  so we can tell how good the algorithm is. For example, is  $T(n) = \Theta(n)$ ? is  $T(n) = \Theta(n^2)$ ? or does it behave differently?

We now start work on this problem. Since the sorting algorithm is recursive (calls itself), we will end up with a recursion. This is a general principle for recursive algorithms. You should see why after the next two paragraphs.

All comparisons are done in Merge(L1,L2). It can be shown that the number of comparisons in Merge is between  $m$  and  $n - 1$ . We take that fact as given.

Three lines of code are important:

$S_1 = \text{Sort}(L_1)$	a recursive call, so it gives us $T(m)$ ;
$S_2 = \text{Sort}(L_2)$	a recursive call, so it gives us $T(n - m)$ ;
$S = \text{Merge}(L_1, L_2)$	where the comparisons are, so it gives us $a_n$ with $m \leq a_n \leq n - 1$ .

We obtain  $T(n) = T(m) + T(n - m) + a_n$  where all we know about  $a_n$  is that it is between  $m$  and  $n - 1$ . What can we do?

Not only is this a type of recursion we haven't seen before, we don't even know the recursion fully since all we have is upper and lower bounds for  $a_n$ . The next theorem solves this problem for us.  $\square$

The following theorem provides an approximate solution to an important class of approximate recursions that arise in divide and conquer algorithms. We'll apply it to merge sorting. In the theorem

- $T(n)$  is the running time for a problem of size  $n$ .
- If the algorithm calls itself at  $w$  places in the code, then the problem is divided into  $w$  smaller problems of the same kind and  $s_1(n), \dots, s_w(n)$  are the sizes of the smaller problems.
- The constant  $c$  measures how much smaller each of these problems is.
- The time needed for the rest of the code is  $a_n$ .

**\*Theorem 8 (Master Theorem for Recursions\*)** Suppose that there are

- (i) numbers  $N, b, w \geq 1$  and  $0 < c < 1$  that do not depend on  $n$
- (ii) a sequence  $a_1, a_2, \dots,$
- (iii) functions  $s_1, s_2, \dots, s_w$ , and  $T$

such that

- (a)  $T(n) > 0$  for all  $n > N$  and  $a_n \geq 0$  for all  $n > N$ ;

---

\* This is not the most general version of the theorem; however, this version is easier to understand and is usually sufficient. For a more general statement and a proof, see any thorough text on the analysis of algorithms.

## Section 4: Rates of Growth and Analysis of Algorithms

- (b)  $T(n) = a_n + T(s_1(n)) + T(s_2(n)) + \cdots + T(s_w(n))$  for all  $n > N$ ;
- (c)  $a_n$  is  $\Theta(n^b)$  (If  $a_n = 0$  for all large  $n$ , set  $b = -\infty$ .);
- (d)  $|s_i(n) - cn|$  is  $O(1)$  for  $i = 1, 2, \dots, w$ .

Let  $d = -\log(w)/\log(c)$ . Then

$$T(n) \text{ is } \begin{cases} \Theta(n^d) & \text{if } b < d, \\ \Theta(n^d \log n) & \text{if } b = d, \\ \Theta(n^b) & \text{if } b > d. \end{cases}$$

Note that  $b = 0$  corresponds to  $a_n$  being in  $\Theta(1)$  since  $n^0 = 1$ . In other words,  $a_n$  is bounded by nonzero constants for all large  $n$ :  $0 < C_1 \leq a_n \leq C_2$ .

Let's apply the theorem to our recursion for merge sorting:

$$T(n) = a_n + T(s_1(n)) + T(s_2(n))$$

where

$$s_1(n) = \lfloor n/2 \rfloor, \quad s_2(n) = \lfloor n - n/2 \rfloor \quad \text{and} \quad s_1(n) \leq a_n \leq n - 1.$$

Note that  $s_1(n)$  and  $s_2(n)$  differ from  $n/2$  by at most  $1/2$  and that  $a_n = \Theta(n)$ . Thus we can apply the theorem with  $w = 2$ ,  $b = 1$  and  $c = 1/2$ . We have

$$d = -\log(2)/\log(1/2) = \log(2)/\log(2) = 1.$$

Since  $b = d = 1$ , we conclude that  $T(n)$  is  $\Theta(n \log n)$ .

How do we use the theorem on divide and conquer algorithms? First, we must find a parameter  $n$  that measures the size of the problem; for example, the length of a list to be sorted, the degree of polynomials that we want to multiply, the number of vertices in a graph that we want to study. Then use the interpretation of the various parameters that was given just before the theorem.

Our final example is more difficult because the algorithm that we study is more complicated. It was believed for some time that the quickest way to multiply polynomials was the “obvious” way that is taught when polynomials are first studied. That is not true. The next example contains an algorithm for faster multiplication of polynomials. There are also faster algorithms for multiplying matrices.

**\*Example 28 (Recursive multiplication of polynomials)** Suppose we want to multiply two polynomials of degree at most  $n$ , say

$$P(x) = p_0 + p_1x + \cdots + p_nx^n \quad \text{and} \quad Q(x) = q_0 + q_1x + \cdots + q_nx^n.$$

The natural way to do this is to use the distributive law to generate  $(n + 1)^2$  products  $p_0q_0, p_0q_1x, p_0q_2x^2, \dots, p_nq_nx^{2n}$  and then collect the terms that have the same powers of  $x$ . This involves  $(n + 1)^2$  multiplications of coefficients and, it can be shown,  $n^2$  additions of coefficients. Thus, the amount of work is  $\Theta(n^2)$ . Unless we expect  $P(x)$  or  $Q(x)$  to have

## Basic Concepts in Graph Theory

some coefficients that are zero, this seems to be best we can do. Not so! We now present and analyze a faster recursive algorithm.

The algorithm depends on the following identity which you should verify by checking the algebra.

*Identity:* If  $P_L(x)$ ,  $P_H(x)$ ,  $Q_L(x)$  and  $Q_H(x)$  are polynomials, then

$$(P_L(x) + P_H(x)x^m)(Q_L(x) + Q_H(x)x^m) = A(x) + (C(x) - A(x) - B(x))x^m + B(x)x^{2m}$$

where

$$A(x) = P_L(x)Q_L(x), \quad B(x) = P_H(x)Q_H(x),$$

and

$$C(x) = (P_L(x) + P_H(x))(Q_L(x) + Q_H(x))$$

We can think of this identity as telling us how to multiply two polynomials  $P(x)$  and  $Q(x)$  by splitting them into lower degree terms ( $P_L(x)$  and  $Q_L(x)$ ) and higher degree terms ( $P_H(x)x^m$  and  $Q_H(x)x^m$ ):

$$P(x) = D(P_L(x) + P_H(x)x^m) \quad \text{and} \quad Q(x) = Q_L(x) + Q_H(x)x^m.$$

The identity requires three polynomial multiplications to compute  $A(x)$ ,  $B(x)$  and  $C(x)$ . This leads naturally to two questions:

- Haven't things gotten worse — three polynomial multiplications instead of just one? No. The three multiplications involve polynomials of much lower degrees. We will see that this leads to a gain in speed.
- How should we do these three polynomial multiplications? Apply the identity to each of them. In other words, design a recursive algorithm. We do that now.

Here is the algorithm for multiplying two polynomials  $P(x) = p_0 + p_1x + \dots + p_nx^n$  and  $Q(x) = q_0 + q_1x + \dots + q_nx^n$  of degree at most  $n$ .

```

MULT( $P(x), Q(x), n$ )
  If ( $n=0$ ) Return  $p_0q_0$ 
  Else
    Let  $m = n/2$  rounded up.
     $P_L(x) = p_0 + p_1x + \dots + p_{m-1}x^{m-1}$ 
     $P_H(x) = p_m + p_{m+1}x + \dots + p_nx^{n-m}$ 
     $Q_L(x) = q_0 + q_1x + \dots + q_{m-1}x^{m-1}$ 
     $Q_H(x) = q_m + q_{m+1}x + \dots + q_nx^{n-m}$ 
     $A(x) = \text{MULT}(P_L(x), Q_L(x), m - 1)$ 
     $B(x) = \text{MULT}(P_H(x), Q_H(x), n - m)$ 
     $C(x) = \text{MULT}(P_L(x) + P_H(x), Q_L(x) + Q_H(x), n - m)$ 
     $D(x) = A(x) + (C(x) - A(x) - B(x))x^m + B(x)x^{2m}$ 
    Return  $D(x)$ 
  End if
End
```

As is commonly done, we imagine a polynomial stored as a vector of coefficients. The amount of work required is then the number of times we have two multiply or add two

## Section 4: Rates of Growth and Analysis of Algorithms

coefficients. For simplicity, we just count multiplications. Let that number be  $T(n)$ . You should be able to see that  $T(0) = 1$  and

$$T(n) = T(m - 1) + T(n - m) + T(n - m) \quad \text{for } n > 0.$$

We can write this as

$$T(n) = T(m - 1) + T(n - m) + T(n - m) + a_n, \quad a_0 = 1 \text{ and } a_n = 0 \text{ for } n > 0.$$

Note that, since both  $m - 1$  and  $n - m$  differ from  $n/2$  by at most 1,  $w = 3$  and  $c = 1/2$ . Also  $b = -\infty$ .

We have  $d = \log 3 / \log 2 > b$ . Thus  $T(n)$  is  $\Theta(n^{\log 3 / \log 2})$ . Since  $\log 3 / \log 2$  is about 1.6 which is less than 2, this is less work than the straightforward method when  $n$  is large enough. (Recall that the work there was in  $\Theta(n^2)$ .)  $\square$

---

### Exercises for Section 4

- 4.1.** We have three algorithms for solving a problem for graphs. Suppose algorithm  $A$  takes  $n^2$  milliseconds to run on a graph with  $n$  vertices, algorithm  $B$  takes  $100n$  milliseconds and algorithm  $C$  takes  $100(2^{n/10} - 1)$  milliseconds.
- (a) Compute the running times for the three algorithms with  $n = 5, 10, 30, 100$  and 300. Which algorithm is fastest in each case? slowest?
  - (b) Which algorithm is fastest for all very large values of  $n$ ? Which is slowest?

- 4.2.** Let  $p(x)$  be a polynomial of degree  $k$  with positive leading coefficient and suppose that  $a > 1$ . Prove the following.

- (a)  $\Theta(p(n))$  is  $\Theta(n^k)$ .
- (b)  $O(p(n))$  is  $O(n^k)$ .
- (c)  $\lim_{n \rightarrow \infty} p(n)/a^n = 0$ . (Also, what does this say about the speed of a polynomial time algorithm versus one which takes exponential time?)
- (d) Unless  $p(x) = p_1x^k + p_2$  for some  $p_1$  and  $p_2$ , there is no  $C$  such that  $a^{p(n)}$  is  $\Theta(a^{Cn^k})$ .

- 4.3.** In each case, prove that  $g(n)$  is  $\Theta(f(n))$  using the definition of “ $g$  is  $\Theta(f)$ ”. (See Definition 21.)

- (a)  $g(n) = n^3 + 5n^2 + 10$ ,  $f(n) = 20n^3$ .
- (b)  $g(n) = n^2 + 5n^2 + 10$ ,  $f(n) = 200n^2$

- 4.4.** In each case, show that the given series has the indicated property.

## Basic Concepts in Graph Theory

- (a)  $\sum_{i=1}^n i^2$  is  $\Theta(n^3)$ .
- (b)  $\sum_{i=1}^n i^3$  is  $\Theta(n^4)$ .
- (c)  $\sum_{i=1}^n i^{1/2}$  is  $\Theta(n^{3/2})$ .

**4.5.** Show each of the following

- (a)  $\sum_{i=1}^n i^{-1}$  is  $\Theta(\log_b(n))$  for any base  $b > 1$ .
- (b)  $\log_b(n!)$  is  $O(n \log_b(n))$  for any base  $b > 1$ .
- (c)  $n!$  is  $\Theta((n/e)^{n+1/2})$ .

**\*4.6.** The following algorithm multiplies two  $n \times n$  matrices  $A$  and  $B$  and puts the answer in  $C$ . Let  $T(n)$  be the running time of the algorithm. Find a simple function  $f(n)$  so that  $T(n) = \Theta(f(n))$ .

```

MATRIXMULT(n,A,B,C)
  For i=1,...,n
    For j=1,...,n
      C(i,j)=0
      For k=1,...,n
        C(i,j) = C(i,j) + A(i,k)*B(k,j)
      End for
    End for
  End for
End

```

**\*4.7.** The following algorithm computes  $x^n$  for  $n$  a positive integer, where  $x$  is a complicated object (e.g., a large matrix).  $MULT(x, y)$  is a procedure that multiplies two such objects. Let  $T(n)$  be the number of times  $MULT$  is called. Find a simple function  $f(n)$  so that  $T(n) = \Theta(f(n))$ .

```

POW(x,n)
  If (n=1) Return x
  Else
    Let q be n/2 rounded down and r = n - 2q.
    y = MULT(x, x)
    z = POW(y, q)
    If (r=0) Return z
    Else
      w = MULT(x, z)
      Return w
    End if
  End if
End

```

## Multiple Choice Questions for Review

Some of the following questions assume that you have done the exercises.

1. Indicate which, if any, of the following five graphs  $G = (V, E, \phi)$ ,  $|V| = 5$ , is not isomorphic to any of the other four.

$$(a) \phi = \begin{pmatrix} A & B & C & D & E & F \\ \{1,3\} & \{2,4\} & \{1,2\} & \{2,3\} & \{3,5\} & \{4,5\} \end{pmatrix}$$

$$(b) \phi = \begin{pmatrix} f & b & c & d & e & a \\ \{1,2\} & \{1,2\} & \{2,3\} & \{3,4\} & \{3,4\} & \{4,5\} \end{pmatrix}$$

$$(c) \phi = \begin{pmatrix} b & f & e & d & c & a \\ \{4,5\} & \{1,3\} & \{1,3\} & \{2,3\} & \{2,4\} & \{4,5\} \end{pmatrix}$$

$$(d) \phi = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 \\ \{1,2\} & \{2,3\} & \{2,3\} & \{3,4\} & \{4,5\} & \{4,5\} \end{pmatrix}$$

$$(e) \phi = \begin{pmatrix} b & a & e & d & c & f \\ \{4,5\} & \{1,3\} & \{1,3\} & \{2,3\} & \{2,5\} & \{4,5\} \end{pmatrix}$$

2. Indicate which, if any, of the following five graphs  $G = (V, E, \phi)$ ,  $|V| = 5$ , is not connected.

$$(a) \phi = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 \\ \{1,2\} & \{1,2\} & \{2,3\} & \{3,4\} & \{1,5\} & \{1,5\} \end{pmatrix}$$

$$(b) \phi = \begin{pmatrix} b & a & e & d & c & f \\ \{4,5\} & \{1,3\} & \{1,3\} & \{2,3\} & \{2,5\} & \{4,5\} \end{pmatrix}$$

$$(c) \phi = \begin{pmatrix} b & f & e & d & c & a \\ \{4,5\} & \{1,3\} & \{1,3\} & \{2,3\} & \{2,4\} & \{4,5\} \end{pmatrix}$$

$$(d) \phi = \begin{pmatrix} a & b & c & d & e & f \\ \{1,2\} & \{2,3\} & \{1,2\} & \{2,3\} & \{3,4\} & \{1,5\} \end{pmatrix}$$

$$(e) \phi = \begin{pmatrix} a & b & c & d & e & f \\ \{1,2\} & \{2,3\} & \{1,2\} & \{1,3\} & \{2,3\} & \{4,5\} \end{pmatrix}$$

3. Indicate which, if any, of the following five graphs  $G = (V, E, \phi)$ ,  $|V| = 5$ , have an Eulerian circuit.

$$(a) \phi = \begin{pmatrix} F & B & C & D & E & A \\ \{1,2\} & \{1,2\} & \{2,3\} & \{3,4\} & \{4,5\} & \{4,5\} \end{pmatrix}$$

$$(b) \phi = \begin{pmatrix} b & f & e & d & c & a \\ \{4,5\} & \{1,3\} & \{1,3\} & \{2,3\} & \{2,4\} & \{4,5\} \end{pmatrix}$$

$$(c) \phi = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 \\ \{1,2\} & \{1,2\} & \{2,3\} & \{3,4\} & \{4,5\} & \{4,5\} \end{pmatrix}$$

$$(d) \phi = \begin{pmatrix} b & a & e & d & c & f \\ \{4,5\} & \{1,3\} & \{1,3\} & \{2,3\} & \{2,5\} & \{4,5\} \end{pmatrix}$$

$$(e) \phi = \begin{pmatrix} a & b & c & d & e & f \\ \{1,3\} & \{3,4\} & \{1,2\} & \{2,3\} & \{3,5\} & \{4,5\} \end{pmatrix}$$

4. A graph with  $V = \{1, 2, 3, 4\}$  is described by  $\phi = \begin{pmatrix} a & b & c & d & e & f \\ \{1,2\} & \{1,2\} & \{1,4\} & \{2,3\} & \{3,4\} & \{3,4\} \end{pmatrix}$ . How many Hamiltonian cycles does it have?

- (a) 1      (b) 2      (c) 4      (d) 16      (e) 32

## Basic Concepts in Graph Theory

5. A graph with  $V = \{1, 2, 3, 4\}$  is described by  $\phi = \begin{pmatrix} a & b & c & d & e & f \\ \{1,2\} & \{1,2\} & \{1,4\} & \{2,3\} & \{3,4\} & \{3,4\} \end{pmatrix}$ . It has weights on its edges given by  $\lambda = \begin{pmatrix} a & b & c & d & e & f \\ 3 & 2 & 1 & 2 & 4 & 2 \end{pmatrix}$ . How many minimum spanning trees does it have?

(a) 2      (b) 3      (c) 4      (d) 5      (e) 6

6. Define an RP-tree by the parent-child adjacency lists as follows:

(i) Root B: J, H, K;   (ii) H: P, Q, R;   (iii) Q: S, T;   (iv) K: L, M, N.

The postorder vertex sequence of this tree is

- (a) J, P, S, T, Q, R, H, L, M, N, K, B.
- (b) P, S, T, J, Q, R, H, L, M, N, K, B.
- (c) P, S, T, Q, R, H, L, M, N, K, J, B.
- (d) P, S, T, Q, R, J, H, L, M, N, K, B.
- (e) S, T, Q, J, P, R, H, L, M, N, K, B.

7. Define an RP-tree by the parent-child adjacency lists as follows:

(i) Root B: J, H, K;   (ii) J: P, Q, R;   (iii) Q: S, T;   (iv) K: L, M, N.

The preorder vertex sequence of this tree is

- (a) B, J, H, K, P, Q, R, L, M, N, S, T.
- (b) B, J, P, Q, S, T, R, H, K, L, M, N.
- (c) B, J, P, Q, S, T, R, H, L, M, N, K.
- (d) B, J, Q, P, S, T, R, H, L, M, N, K.
- (e) B, J, Q, S, T, P, R, H, K, L, M, N.

8. For which of the following does there exist a graph  $G = (V, E, \phi)$  satisfying the specified conditions?

- (a) A tree with 9 vertices and the sum of the degrees of all the vertices 18.
- (b) A graph with 5 components 12 vertices and 7 edges.
- (c) A graph with 5 components 30 vertices and 24 edges.
- (d) A graph with 9 vertices, 9 edges, and no cycles.
- (e) A connected graph with 12 edges 5 vertices and fewer than 8 cycles.

9. For which of the following does there exist a simple graph  $G = (V, E)$  satisfying the specified conditions?

- (a) It has 3 components 20 vertices and 16 edges.
- (b) It has 6 vertices, 11 edges, and more than one component.

## Review Questions

- (c) It is connected and has 10 edges 5 vertices and fewer than 6 cycles.  
(d) It has 7 vertices, 10 edges, and more than two components.  
(e) It has 8 vertices, 8 edges, and no cycles.
- 10.** For which of the following does there exist a tree satisfying the specified constraints?
- (a) A binary tree with 65 leaves and height 6.  
(b) A binary tree with 33 leaves and height 5.  
(c) A full binary tree with height 5 and 64 total vertices.  
(d) A full binary tree with 23 leaves and height 23.  
(e) A rooted tree of height 3, every vertex has at most 3 children. There are 40 total vertices.
- 11.** For which of the following does there exist a tree satisfying the specified constraints?
- (a) A full binary tree with 31 leaves, each leaf of height 5.  
(b) A rooted tree of height 3 where every vertex has at most 3 children and there are 41 total vertices.  
(c) A full binary tree with 11 vertices and height 6.  
(d) A binary tree with 2 leaves and height 100.  
(e) A full binary tree with 20 vertices.
- 12.** The number of simple digraphs with  $|V| = 3$  is
- (a)  $2^9$     (b)  $2^8$     (c)  $2^7$     (d)  $2^6$     (e)  $2^5$
- 13.** The number of simple digraphs with  $|V| = 3$  and exactly 3 edges is
- (a) 92    (b) 88    (c) 80    (d) 84    (e) 76
- 14.** The number of oriented simple graphs with  $|V| = 3$  is
- (a) 27    (b) 24    (c) 21    (d) 18    (e) 15
- 15.** The number of oriented simple graphs with  $|V| = 4$  and 2 edges is
- (a) 40    (b) 50    (c) 60    (d) 70    (e) 80
- 16.** In each case the depth-first sequence of an ordered rooted spanning tree for a graph  $G$  is given. Also given are the non-tree edges of  $G$ . Which of these spanning trees is a depth-first spanning tree?
- (a) 123242151 and  $\{3, 4\}, \{1, 4\}$   
(b) 123242151 and  $\{4, 5\}, \{1, 3\}$   
(c) 123245421 and  $\{2, 5\}, \{1, 4\}$   
(d) 123245421 and  $\{3, 4\}, \{1, 4\}$   
(e) 123245421 and  $\{3, 5\}, \{1, 4\}$

## Basic Concepts in Graph Theory

- 17.**  $\sum_{i=1}^n i^{-1/2}$  is  
 (a)  $\Theta((\ln(n))^{1/2})$       (b)  $\Theta(\ln(n))$       (c)  $\Theta(n^{1/2})$       (d)  $\Theta(n^{3/2})$       (e)  $\Theta(n^2)$
- 18.** Compute the total number of biconponents in all of the following three simple graphs,  $G = (V, E)$  with  $|V| = 5$ . For each graph the edge sets are as follows:
- $$E = \{\{1, 2\}, \{2, 3\}, \{3, 4\}, \{4, 5\}, \{1, 3\}, \{1, 5\}, \{3, 5\}\}$$
- $$E = \{\{1, 2\}, \{2, 3\}, \{3, 4\}, \{4, 5\}, \{1, 3\}\}$$
- $$E = \{\{1, 2\}, \{2, 3\}, \{4, 5\}, \{1, 3\}\}$$
- (a) 4      (b) 5      (c) 6      (d) 7      (e) 8
- 19.** Let  $b > 1$ . Then  $\log_b((n^2)!)$  is
- (a)  $\Theta(\log_b(n!))$   
 (b)  $\Theta(\log_b(2 n!))$   
 (c)  $\Theta(n \log_b(n))$   
 (d)  $\Theta(n^2 \log_b(n))$   
 (e)  $\Theta(n \log_b(n^2))$
- 20.** What is the total number of additions and multiplications in the following code?

```

s := 0
for i := 1 to n
    s := s + i
    for j := 1 to i
        s := s + j*i
    next j
next i
s := s+10
    
```

- (a)  $n$       (b)  $n^2$       (c)  $n^2 + 2n$       (d)  $n(n + 1)$       (e)  $(n + 1)^2$

**Answers:** **1** (a), **2** (e), **3** (e), **4** (c), **5** (b), **6** (a), **7** (b), **8** (b), **9** (d), **10** (e), **11** (d), **12** (a), **13** (d), **14** (a), **15** (c), **16** (c), **17** (c), **18** (c), **19** (d), **20** (e).

---

# An Introduction to Combinatorics and Graph Theory

---

David Guichard



This work is licensed under the Creative Commons Attribution-NonCommercial-ShareAlike License. To view a copy of this license, visit <http://creativecommons.org/licenses/by-nc-sa/3.0/> or send a letter to Creative Commons, 543 Howard Street, 5th Floor, San Francisco, California, 94105, USA. If you distribute this work or a derivative, include the history of the document.

This copy of the text was compiled from source at 8:50 on 5/7/2018.

We will be glad to receive corrections and suggestions for improvement at [guichard@whitman.edu](mailto:guichard@whitman.edu).

# Contents

## 1

---

Fundamentals	7
--------------	---

1.1 Examples . . . . .	8
1.2 Combinations and permutations . . . . .	11
1.3 Binomial coefficients . . . . .	16
1.4 Bell numbers . . . . .	22
1.5 Choice with repetition . . . . .	27
1.6 The Pigeonhole Principle . . . . .	32
1.7 Sperner's Theorem . . . . .	36
1.8 Stirling numbers . . . . .	39

## 2

---

Inclusion-Exclusion	45
---------------------	----

2.1 The Inclusion-Exclusion Formula . . . . .	45
2.2 Forbidden Position Permutations . . . . .	48

## 4 Contents

### 3

---

#### Generating Functions 53

<b>3.1</b>	Newton's Binomial Theorem . . . . .	53
<b>3.2</b>	Exponential Generating Functions . . . . .	56
<b>3.3</b>	Partitions of Integers . . . . .	59
<b>3.4</b>	Recurrence Relations . . . . .	62
<b>3.5</b>	Catalan Numbers . . . . .	66

### 4

---

#### Systems of Distinct Representatives 71

<b>4.1</b>	Existence of SDRs . . . . .	72
<b>4.2</b>	Partial SDRs . . . . .	74
<b>4.3</b>	Latin Squares . . . . .	76
<b>4.4</b>	Introduction to Graph Theory . . . . .	83
<b>4.5</b>	Matchings . . . . .	84

### 5

---

#### Graph Theory 91

<b>5.1</b>	The Basics . . . . .	91
<b>5.2</b>	Euler Circuits and Walks . . . . .	96
<b>5.3</b>	Hamilton Cycles and Paths . . . . .	100
<b>5.4</b>	Bipartite Graphs . . . . .	103
<b>5.5</b>	Trees . . . . .	105
<b>5.6</b>	Optimal Spanning Trees . . . . .	108
<b>5.7</b>	Connectivity . . . . .	110
<b>5.8</b>	Graph Coloring . . . . .	117
<b>5.9</b>	The Chromatic Polynomial . . . . .	124
<b>5.10</b>	Coloring Planar Graphs . . . . .	125
<b>5.11</b>	Directed Graphs . . . . .	129

---

**6**

---

<b>Pólya–Redfield Counting</b>	<b>135</b>
--------------------------------	------------

<b>6.1</b>	Groups of Symmetries . . . . .	<b>137</b>
------------	--------------------------------	------------

<b>6.2</b>	Burnside's Theorem . . . . .	<b>140</b>
------------	------------------------------	------------

<b>6.3</b>	Pólya–Redfield Counting . . . . .	<b>146</b>
------------	-----------------------------------	------------

---

**A**

---

<b>Hints</b>	<b>151</b>
--------------	------------

---

<b>Index</b>	<b>153</b>
--------------	------------

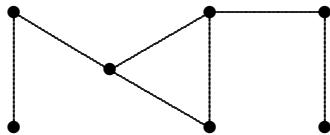


# 1

## Fundamentals

Combinatorics is often described briefly as being about counting, and indeed counting is a large part of combinatorics. As the name suggests, however, it is broader than this: it is about combining things. Questions that arise include counting problems: “How many ways can these elements be combined?” But there are other questions, such as whether a certain combination is possible, or what combination is the “best” in some sense. We will see all of these, though counting plays a particularly large role.

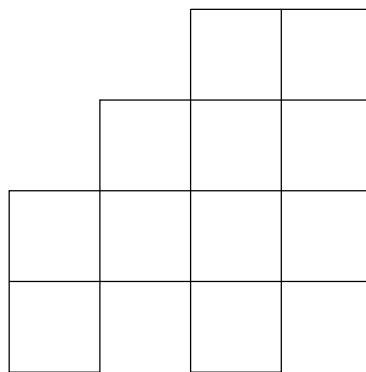
Graph theory is concerned with various types of networks, or really models of networks called graphs. These are not the graphs of analytic geometry, but what are often described as “points connected by lines”, for example:



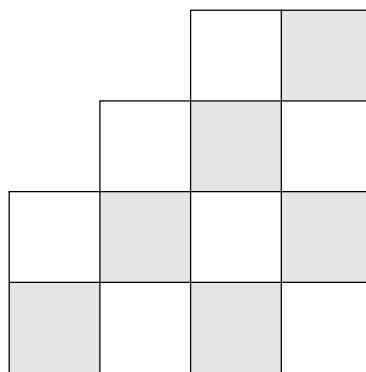
The preferred terminology is **vertex** for a point and **edge** for a line. The lines need not be straight lines, and in fact the actual definition of a graph is not a geometric definition. The figure above is simply a visualization of a graph; the graph is a more abstract object, consisting of seven vertices, which we might name  $\{v_1, \dots, v_7\}$ , and the collection of pairs of vertices that are connected; for a suitable assignment of names  $v_i$  to the points in the diagram, the edges could be represented as  $\{v_1, v_2\}, \{v_2, v_3\}, \{v_3, v_4\}, \{v_3, v_5\}, \{v_4, v_5\}, \{v_5, v_6\}, \{v_6, v_7\}$ .

## 1.1 EXAMPLES

Suppose we have a chess board, and a collection of tiles, like dominoes, each of which is the size of two squares on the chess board. Can the chess board be covered by the dominoes? First we need to be clear on the rules: the board is covered if the dominoes are laid down so that each covers exactly two squares of the board; no dominoes overlap; and every square is covered. The answer is easy: simply by laying out 32 dominoes in rows, the board can be covered. To make the problem more interesting, we allow the board to be rectangular of any size, and we allow some squares to be removed from the board. What can be say about whether the remaining board can be covered? This is such a board, for example:

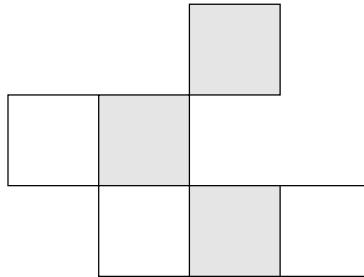


What can we say? Here is an easy observation: each domino must cover two squares, so the total number of squares must be even; the board above has an even number of squares. Is that enough? It is not too hard to convince yourself that this board cannot be covered; is there some general principle at work? Suppose we redraw the board to emphasize that it really is part of a chess board:

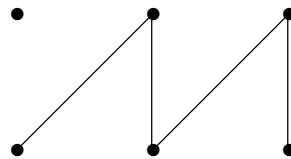


Aha! Every tile must cover one white and one gray square, but there are four of the former and six of the latter, so it is impossible. Now do we have the whole picture? No;

for example:

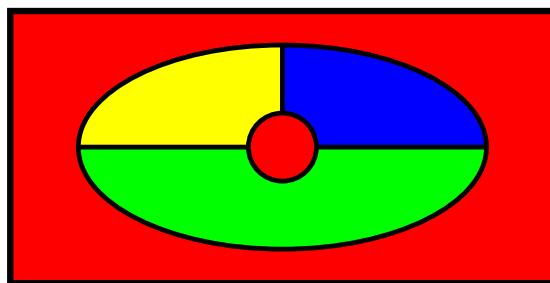


The gray square at the upper right clearly cannot be covered. Unfortunately it is not easy to state a condition that fully characterizes the boards that can be covered; we will see this problem again. Let us note, however, that this problem can also be represented as a graph problem. We introduce a vertex corresponding to each square, and connect two vertices by an edge if their associated squares can be covered by a single domino; here is the previous board:



Here the top row of vertices represents the gray squares, the bottom row the white squares. A domino now corresponds to an edge; a covering by dominoes corresponds to a collection of edges that share no endpoints and that are **incident** with (that is, touch) all six vertices. Since no edge is incident with the top left vertex, there is no cover.

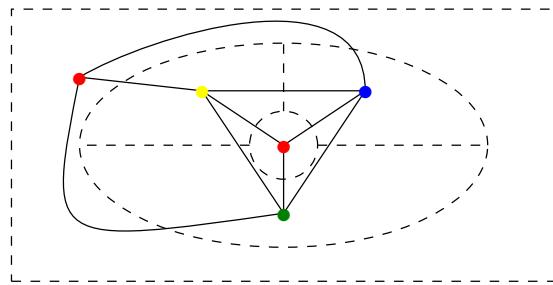
Perhaps the most famous problem in graph theory concerns map coloring: Given a map of some countries, how many colors are required to color the map so that countries sharing a border get different colors? It was long conjectured that any map could be colored with four colors, and this was finally proved in 1976. Here is an example of a small map, colored with four colors:



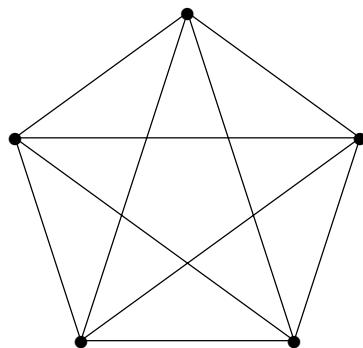
Typically this problem is turned into a graph theory problem. Suppose we add to each country a capital, and connect capitals across common boundaries. Coloring the capitals so

## 10 Chapter 1 Fundamentals

that no two connected capitals share a color is clearly the same problem. For the previous map:



Any graph produced in this way will have an important property: it can be drawn so that no edges cross each other; this is a **planar** graph. Non-planar graphs can require more than four colors, for example this graph:

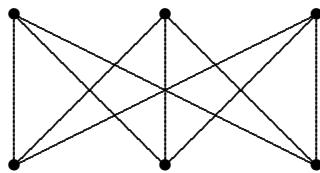


This is called the **complete graph** on five vertices, denoted  $K_5$ ; in a complete graph, each vertex is connected to each of the others. Here only the “fat” dots represent vertices; intersections of edges at other points are not vertices. A few minutes spent trying should convince you that this graph cannot be drawn so that its edges don’t cross, though the number of edge crossings can be reduced.

### *Exercises 1.1.*

1. Explain why an  $m \times n$  board can be covered if either  $m$  or  $n$  is even. Explain why it cannot be covered if both  $m$  and  $n$  are odd.
2. Suppose two diagonally opposite corners of an ordinary  $8 \times 8$  board are removed. Can the resulting board be covered?
3. Suppose that  $m$  and  $n$  are both odd. On an  $m \times n$  board, colored as usual, all four corners will be the same color, say white. Suppose one white square is removed from any location on the board. Show that the resulting board can be covered.
4. Suppose that one corner of an  $8 \times 8$  board is removed. Can the remainder be covered by  $1 \times 3$  tiles? Show a tiling or prove that it cannot be done.

5. Suppose the square in row 3, column 3 of an  $8 \times 8$  board is removed. Can the remainder be covered by  $1 \times 3$  tiles? Show a tiling or prove that it cannot be done.
6. Remove two diagonally opposite corners of an  $m \times n$  board, where  $m$  is odd and  $n$  is even. Show that the remainder can be covered with dominoes.
7. Suppose one white and one black square are removed from an  $n \times n$  board,  $n$  even. Show that the remainder can be covered by dominoes.
8. Suppose an  $n \times n$  board,  $n$  even, is covered with dominoes. Show that the number of horizontal dominoes with a white square under the left end is equal to the number of horizontal dominoes with a black square under the left end.
9. In the complete graph on five vertices shown above, there are five pairs of edges that cross. Draw this graph so that only one pair of edges cross. Remember that “edges” do not have to be straight lines.
10. The **complete bipartite graph**  $K_{3,3}$  consists of two groups of three vertices each, with all possible edges between the groups and no other edges:



Draw this graph with only one crossing.

## 1.2 COMBINATIONS AND PERMUTATIONS

We turn first to *counting*. While this sounds simple, perhaps too simple to study, it is not. When we speak of counting, it is shorthand for determining the size of a set, or more often, the sizes of many sets, all with something in common, but different sizes depending on one or more parameters. For example: how many outcomes are possible when a die is rolled? Two dice?  $n$  dice? As stated, this is ambiguous: what do we mean by “outcome”? Suppose we roll two dice, say a red die and a green die. Is “red two, green three” a different outcome than “red three, green two”? If yes, we are counting the number of possible “physical” outcomes, namely 36. If no, there are 21. We might even be interested simply in the possible totals, in which case there are 11 outcomes.

Even the quite simple first interpretation relies on some degree of knowledge about counting; we first make two simple facts explicit. In terms of set sizes, suppose we know that set  $A$  has size  $m$  and set  $B$  has size  $n$ . What is the size of  $A$  and  $B$  together, that is, the size of  $A \cup B$ ? If we know that  $A$  and  $B$  have no elements in common, then the size  $A \cup B$  is  $m + n$ ; if they do have elements in common, we need more information. A simple but typical problem of this type: if we roll two dice, how many ways are there to get either 7 or 11? Since there are 6 ways to get 7 and two ways to get 11, the answer is  $6 + 2 = 8$ . Though this principle is simple, it is easy to forget the requirement that the two sets be

## 12 Chapter 1 Fundamentals

disjoint, and hence to use it when the circumstances are otherwise. This principle is often called the **addition principle**.

This principle can be generalized: if sets  $A_1$  through  $A_n$  are pairwise disjoint and have sizes  $m_1, \dots, m_n$ , then the size of  $A_1 \cup \dots \cup A_n = \sum_{i=1}^n m_i$ . This can be proved by a simple induction argument.

Why do we know, without listing them all, that there are 36 outcomes when two dice are rolled? We can view the outcomes as two separate outcomes, that is, the outcome of rolling die number one and the outcome of rolling die number two. For each of 6 outcomes for the first die the second die may have any of 6 outcomes, so the total is  $6 + 6 + 6 + 6 + 6 + 6 = 36$ , or more compactly,  $6 \cdot 6 = 36$ . Note that we are really using the addition principle here: set  $A_1$  is all pairs  $(1, x)$ , set  $A_2$  is all pairs  $(2, x)$ , and so on. This is somewhat more subtle than is first apparent. In this simple example, the outcomes of die number two have nothing to do with the outcomes of die number one. Here's a slightly more complicated example: how many ways are there to roll two dice so that the two dice don't match? That is, we rule out 1-1, 2-2, and so on. Here for each possible value on die number one, there are five possible values for die number two, but they are a different five values for each value on die number one. Still, because all are the same, the result is  $5 + 5 + 5 + 5 + 5 = 30$ , or  $6 \cdot 5 = 30$ . In general, then, if there are  $m$  possibilities for one event, and  $n$  for a second event, the number of possible outcomes for both events together is  $m \cdot n$ . This is often called the **multiplication principle**.

In general, if  $n$  events have  $m_i$  possible outcomes, for  $i = 1, \dots, n$ , where each  $m_i$  is unaffected by the outcomes of other events, then the number of possible outcomes overall is  $\prod_{i=1}^n m_i$ . This too can be proved by induction.

**EXAMPLE 1.2.1** How many outcomes are possible when three dice are rolled, if no two of them may be the same? The first two dice together have  $6 \cdot 5 = 30$  possible outcomes, from above. For each of these 30 outcomes, there are four possible outcomes for the third die, so the total number of outcomes is  $30 \cdot 4 = 6 \cdot 5 \cdot 4 = 120$ . (Note that we consider the dice to be distinguishable, that is, a roll of 6, 4, 1 is different than 4, 6, 1, because the first and second dice are different in the two rolls, even though the numbers as a set are the same.) □

**EXAMPLE 1.2.2** Suppose blocks numbered 1 through  $n$  are in a barrel; we pull out  $k$  of them, placing them in a line as we do. How many outcomes are possible? That is, how many different arrangements of  $k$  blocks might we see?

This is essentially the same as the previous example: there are  $k$  “spots” to be filled by blocks. Any of the  $n$  blocks might appear first in the line; then any of the remaining  $n - 1$  might appear next, and so on. The number of outcomes is thus  $n(n-1)(n-2) \cdots (n-k+1)$ ,

by the multiplication principle. In the previous example, the first “spot” was die number one, the second spot was die number two, the third spot die number three, and  $6 \cdot 5 \cdot 4 = 6(6 - 1)(6 - 2)$ ; notice that  $6 - 2 = 6 - 3 + 1$ .  $\square$

This is quite a general sort of problem:

**DEFINITION 1.2.3** The number of permutations of  $n$  things taken  $k$  at a time is

$$P(n, k) = n(n - 1)(n - 2) \cdots (n - k + 1) = \frac{n!}{(n - k)!}.$$

$\square$

A permutation of some objects is a particular linear ordering of the objects;  $P(n, k)$  in effect counts two things simultaneously: the number of ways to choose and order  $k$  out of  $n$  objects. A useful special case is  $k = n$ , in which we are simply counting the number of ways to order all  $n$  objects. This is  $n(n - 1) \cdots (n - n + 1) = n!$ . Note that the second form of  $P(n, k)$  from the definition gives

$$\frac{n!}{(n - n)!} = \frac{n!}{0!}.$$

This is correct only if  $0! = 1$ , so we adopt the standard convention that this is true, that is, we *define*  $0!$  to be 1.

Suppose we want to count only the number of ways to choose  $k$  items out of  $n$ , that is, we don’t care about order. In example 1.2.1, we counted the number of rolls of three dice with different numbers showing. The dice were distinguishable, or in a particular order: a first die, a second, and a third. Now we want to count simply how many combinations of numbers there are, with 6, 4, 1 now counting as the same combination as 4, 6, 1.

**EXAMPLE 1.2.4** Suppose we were to list all 120 possibilities in example 1.2.1. The list would contain many outcomes that we now wish to count as a single outcome; 6, 4, 1 and 4, 6, 1 would be on the list, but should not be counted separately. How many times will a single outcome appear on the list? This is a permutation problem: there are  $3!$  orders in which 1, 4, 6 can appear, and all 6 of these will be on the list. In fact every outcome will appear on the list 6 times, since every outcome can appear in  $3!$  orders. Hence, the list is too big by a factor of 6; the correct count for the new problem is  $120/6 = 20$ .  $\square$

Following the same reasoning in general, if we have  $n$  objects, the number of ways to choose  $k$  of them is  $P(n, k)/k!$ , as each collection of  $k$  objects will be counted  $k!$  times by  $P(n, k)$ .

**DEFINITION 1.2.5** The number of subsets of size  $k$  of a set of size  $n$  (also called an  $n$ -set) is

$$C(n, k) = \frac{P(n, k)}{k!} = \frac{n!}{k!(n-k)!} = \binom{n}{k}.$$

The notation  $C(n, k)$  is rarely used; instead we use  $\binom{n}{k}$ , pronounced “ $n$  choose  $k$ ”. □

**EXAMPLE 1.2.6** Consider  $n = 0, 1, 2, 3$ . It is easy to list the subsets of a small  $n$ -set; a typical  $n$ -set is  $\{a_1, a_2, \dots, a_n\}$ . A 0-set, namely the empty set, has one subset, the empty set; a 1-set has two subsets, the empty set and  $\{a_1\}$ ; a 2-subset has four subsets,  $\emptyset, \{a_1\}, \{a_2\}, \{a_1, a_2\}$ ; and a 3-subset has eight:  $\emptyset, \{a_1\}, \{a_2\}, \{a_3\}, \{a_1, a_2\}, \{a_1, a_3\}, \{a_2, a_3\}, \{a_1, a_2, a_3\}$ . From these lists it is then easy to compute  $\binom{n}{k}$ :

		$k$			
		0	1	2	3
$n$	0	1			
	1		1	1	
	2		1	2	1
	3		1	3	3

□

You probably recognize these numbers: this is the beginning of **Pascal’s Triangle**. Each entry in Pascal’s triangle is generated by adding two entries from the previous row: the one directly above, and the one above and to the left. This suggests that  $\binom{n}{k} = \binom{n-1}{k-1} + \binom{n-1}{k}$ , and indeed this is true. To make this work out neatly, we adopt the convention that  $\binom{n}{k} = 0$  when  $k < 0$  or  $k > n$ .

**THEOREM 1.2.7**  $\binom{n}{k} = \binom{n-1}{k-1} + \binom{n-1}{k}$ .

**Proof.** A typical  $n$ -set is  $A = \{a_1, \dots, a_n\}$ . We consider two types of subsets: those that contain  $a_n$  and those that do not. If a  $k$ -subset of  $A$  does not contain  $a_n$ , then it is a  $k$ -subset of  $\{a_1, \dots, a_{n-1}\}$ , and there are  $\binom{n-1}{k}$  of these. If it does contain  $a_n$ , then it consists of  $a_n$  and  $k-1$  elements of  $\{a_1, \dots, a_{n-1}\}$ ; since there are  $\binom{n-1}{k-1}$  of these, there are  $\binom{n-1}{k-1}$  subsets of this type. Thus the total number of  $k$ -subsets of  $A$  is  $\binom{n-1}{k-1} + \binom{n-1}{k}$ .

Note that when  $k = 0$ ,  $\binom{n-1}{k-1} = \binom{n-1}{-1} = 0$ , and when  $k = n$ ,  $\binom{n-1}{k} = \binom{n-1}{n} = 0$ , so that  $\binom{n}{0} = \binom{n-1}{0}$  and  $\binom{n}{n} = \binom{n-1}{n-1}$ . These values are the boundary ones in Pascal’s Triangle. ■

Many counting problems rely on the sort of reasoning we have seen. Here are a few variations on the theme.

**EXAMPLE 1.2.8** Six people are to sit at a round table; how many seating arrangements are there?

It is not clear exactly what we mean to count here. If there is a “special seat”, for example, it may matter who ends up in that seat. If this doesn’t matter, we only care about the relative position of each person. Then it may or may not matter whether a certain person is on the left or right of another. So this question can be interpreted in (at least) three ways. Let’s answer them all.

First, if the actual chairs occupied by people matter, then this is exactly the same as lining six people up in a row: 6 choices for seat number one, 5 for seat two, and so on, for a total of  $6!$ . If the chairs don’t matter, then  $6!$  counts the same arrangement too many times, once for each person who might be in seat one. So the total in this case is  $6!/6 = 5!$ . Another approach to this: since the actual seats don’t matter, just put one of the six people in a chair. Then we need to arrange the remaining 5 people in a row, which can be done in  $5!$  ways. Finally, suppose all we care about is who is next to whom, ignoring right and left. Then the previous answer counts each arrangement twice, once for the counterclockwise order and once for clockwise. So the total is  $5!/2 = P(5, 3)$ .  $\square$

We have twice seen a general principle at work: if we can overcount the desired set in such a way that every item gets counted the same number of times, we can get the desired count just by dividing by the common overcount factor. This will continue to be a useful idea. A variation on this theme is to overcount and then *subtract* the amount of overcount.

**EXAMPLE 1.2.9** How many ways are there to line up six people so that a particular pair of people are not adjacent?

Denote the people  $A$  and  $B$ . The total number of orders is  $6!$ , but this counts those orders with  $A$  and  $B$  next to each other. How many of these are there? Think of these two people as a unit; how many ways are there to line up the  $AB$  unit with the other 4 people? We have 5 items, so the answer is  $5!$ . Each of these orders corresponds to two different orders in which  $A$  and  $B$  are adjacent, depending on whether  $A$  or  $B$  is first. So the  $6!$  count is too high by  $2 \cdot 5!$  and the count we seek is  $6! - 2 \cdot 5! = 4 \cdot 5!$ .  $\square$

### Exercises 1.2.

- How many positive factors does  $2 \cdot 3^4 \cdot 7^3 \cdot 11^2 \cdot 47^5$  have? How many does  $p_1^{e_1} p_2^{e_2} \cdots p_n^{e_n}$  have, where the  $p_i$  are distinct primes?
- A poker hand consists of five cards from a standard 52 card deck with four suits and thirteen values in each suit; the order of the cards in a hand is irrelevant. How many hands consist of 2 cards with one value and 3 cards of another value (a full house)? How many consist of 5 cards from the same suit (a flush)?

## 16 Chapter 1 Fundamentals

3. Six men and six women are to be seated around a table, with men and women alternating. The chairs don't matter, only who is next to whom, but right and left are different. How many seating arrangements are possible?
4. Eight people are to be seated around a table; the chairs don't matter, only who is next to whom, but right and left are different. Two people, X and Y, cannot be seated next to each other. How many seating arrangements are possible?
5. In chess, a rook attacks any piece in the same row or column as the rook, provided no other piece is between them. In how many ways can eight indistinguishable rooks be placed on a chess board so that no two attack each other? What about eight indistinguishable rooks on a  $10 \times 10$  board?
6. Suppose that we want to place 8 non-attacking rooks on a chessboard. In how many ways can we do this if the 16 most 'northwest' squares must be empty? How about if only the 4 most 'northwest' squares must be empty?
7. A "legal" sequence of parentheses is one in which the parentheses can be properly matched, like  $()(())$ . It's not hard to see that this is possible precisely when the number of left and right parentheses is the same, and every initial segment of the sequence has at least as many left parentheses as right. For example,  $()\dots$  cannot possibly be extended to a legal sequence. Show that the number of legal sequences of length  $2n$  is  $C_n = \binom{2n}{n} - \binom{2n}{n+1}$ . The numbers  $C_n$  are called the **Catalan numbers**.

## 1.3 BINOMIAL COEFFICIENTS

Recall the appearance of Pascal's Triangle in example 1.2.6. If you have encountered the triangle before, you may know it has many interesting properties. We will explore some of these here.

You may know, for example, that the entries in Pascal's Triangle are the coefficients of the polynomial produced by raising a binomial to an integer power. For example,  $(x+y)^3 = 1 \cdot x^3 + 3 \cdot x^2y + 3 \cdot xy^2 + 1 \cdot y^3$ , and the coefficients 1, 3, 3, 1 form row three of Pascal's Triangle. For this reason the numbers  $\binom{n}{k}$  are usually referred to as the **binomial coefficients**.

### THEOREM 1.3.1 Binomial Theorem

$$(x+y)^n = \binom{n}{0}x^n + \binom{n}{1}x^{n-1}y + \binom{n}{2}x^{n-2}y^2 + \cdots + \binom{n}{n}y^n = \sum_{i=0}^n \binom{n}{i}x^{n-i}y^i$$

**Proof.** We prove this by induction on  $n$ . It is easy to check the first few, say for  $n = 0, 1, 2$ , which form the base case. Now suppose the theorem is true for  $n-1$ , that is,

$$(x+y)^{n-1} = \sum_{i=0}^{n-1} \binom{n-1}{i}x^{n-1-i}y^i.$$

Then

$$(x+y)^n = (x+y)(x+y)^{n-1} = (x+y) \sum_{i=0}^{n-1} \binom{n-1}{i} x^{n-1-i} y^i.$$

Using the distributive property, this becomes

$$\begin{aligned} x \sum_{i=0}^{n-1} \binom{n-1}{i} x^{n-1-i} y^i + y \sum_{i=0}^{n-1} \binom{n-1}{i} x^{n-1-i} y^i \\ = \sum_{i=0}^{n-1} \binom{n-1}{i} x^{n-i} y^i + \sum_{i=0}^{n-1} \binom{n-1}{i} x^{n-1-i} y^{i+1}. \end{aligned}$$

These two sums have much in common, but it is slightly disguised by an “offset”: the first sum starts with an  $x^n y^0$  term and ends with an  $x^1 y^{n-1}$  term, while the corresponding terms in the second sum are  $x^{n-1} y^1$  and  $x^0 y^n$ . Let’s rewrite the second sum so that they match:

$$\begin{aligned} & \sum_{i=0}^{n-1} \binom{n-1}{i} x^{n-i} y^i + \sum_{i=0}^{n-1} \binom{n-1}{i} x^{n-1-i} y^{i+1} \\ &= \sum_{i=0}^{n-1} \binom{n-1}{i} x^{n-i} y^i + \sum_{i=1}^n \binom{n-1}{i-1} x^{n-i} y^i \\ &= \binom{n-1}{0} x^n + \sum_{i=1}^{n-1} \binom{n-1}{i} x^{n-i} y^i + \sum_{i=1}^{n-1} \binom{n-1}{i-1} x^{n-i} y^i + \binom{n-1}{n-1} y^n \\ &= \binom{n-1}{0} x^n + \sum_{i=1}^{n-1} (\binom{n-1}{i} + \binom{n-1}{i-1}) x^{n-i} y^i + \binom{n-1}{n-1} y^n. \end{aligned}$$

Now we can use theorem 1.2.7 to get:

$$\begin{aligned} & \binom{n-1}{0} x^n + \sum_{i=1}^{n-1} (\binom{n-1}{i} + \binom{n-1}{i-1}) x^{n-i} y^i + \binom{n-1}{n-1} y^n. \\ &= \binom{n-1}{0} x^n + \sum_{i=1}^{n-1} \binom{n}{i} x^{n-i} y^i + \binom{n-1}{n-1} y^n. \\ &= \binom{n}{0} x^n + \sum_{i=1}^{n-1} \binom{n}{i} x^{n-i} y^i + \binom{n}{n} y^n \\ &= \sum_{i=0}^n \binom{n}{i} x^{n-i} y^i. \end{aligned}$$

At the next to last step we used the facts that  $\binom{n-1}{0} = \binom{n}{0}$  and  $\binom{n-1}{n-1} = \binom{n}{n}$ . ■

## 18 Chapter 1 Fundamentals

Here is an interesting consequence of this theorem: Consider

$$(x + y)^n = (x + y)(x + y) \cdots (x + y).$$

One way we might think of attempting to multiply this out is this: Go through the  $n$  factors  $(x + y)$  and in each factor choose either the  $x$  or the  $y$ ; at the end, multiply your choices together, getting some term like  $xxxyxy \cdots yx = x^i y^j$ , where of course  $i + j = n$ . If we do this in all possible ways and then collect like terms, we will clearly get

$$\sum_{i=0}^n a_i x^{n-i} y^i.$$

We know that the correct expansion has  $\binom{n}{i} = a_i$ ; is that in fact what we will get by this method? Yes: consider  $x^{n-i} y^i$ . How many times will we get this term using the given method? It will be the number of times we end up with  $i$   $y$ -factors. Since there are  $n$  factors  $(x + y)$ , the number of times we get  $i$   $y$ -factors must be the number of ways to pick  $i$  of the  $(x + y)$  factors to contribute a  $y$ , namely  $\binom{n}{i}$ . This is probably not a useful method in practice, but it is interesting and occasionally useful.

**EXAMPLE 1.3.2** Using this method we might get

$$(x + y)^3 = xxx + xxy + xyx + xyy + yxx + yxy + yyx + yyy$$

which indeed becomes  $x^3 + 3x^2y + 3xy^2 + y^3$  upon collecting like terms.  $\square$

The Binomial Theorem, 1.3.1, can be used to derive many interesting identities. A common way to rewrite it is to substitute  $y = 1$  to get

$$(x + 1)^n = \sum_{i=0}^n \binom{n}{i} x^{n-i}.$$

If we then substitute  $x = 1$  we get

$$2^n = \sum_{i=0}^n \binom{n}{i},$$

that is, row  $n$  of Pascal's Triangle sums to  $2^n$ . This is also easy to understand combinatorially: the sum represents the total number of subsets of an  $n$ -set, since it adds together the numbers of subsets of every possible size. It is easy to see directly that the number of subsets of an  $n$ -set is  $2^n$ : for each element of the set we make a choice, to include or to exclude the element. The total number of ways to make these choices is  $2 \cdot 2 \cdots 2 = 2^n$ , by the multiplication principle.

Suppose now that  $n \geq 1$  and we substitute  $-1$  for  $x$ ; we get

$$(-1 + 1)^n = \sum_{i=0}^n \binom{n}{i} (-1)^{n-i}. \quad (1.3.1)$$

The sum is now an alternating sum: every other term is multiplied by  $-1$ . Since the left hand side is 0, we can rewrite this to get

$$\binom{n}{0} + \binom{n}{2} + \cdots = \binom{n}{1} + \binom{n}{3} + \cdots. \quad (1.3.2)$$

So each of these sums is  $2^{n-1}$ .

Another obvious feature of Pascal's Triangle is symmetry: each row reads the same forwards and backwards. That is, we have:

**THEOREM 1.3.3**  $\binom{n}{i} = \binom{n}{n-i}.$

**Proof.** This is quite easy to see combinatorially: every  $i$ -subset of an  $n$ -set is associated with an  $(n - i)$ -subset. That is, if the  $n$ -set is  $A$ , and if  $B \subseteq A$  has size  $i$ , then the complement of  $B$  has size  $n - i$ . This establishes a 1–1 correspondence between sets of size  $i$  and sets of size  $n - i$ , so the numbers of each are the same. (Of course, if  $i = n - i$ , no proof is required.) ■

Note that this means that the Binomial Theorem, 1.3.1, can also be written as

$$(x + y)^n = \sum_{i=0}^n \binom{n}{n-i} x^{n-i} y^i.$$

or

$$(x + y)^n = \sum_{i=0}^n \binom{n}{i} x^i y^{n-i}.$$

Another striking feature of Pascal's Triangle is that the entries across a row are strictly increasing to the middle of the row, and then strictly decreasing. Since we already know that the rows are symmetric, the first part of this implies the second.

**THEOREM 1.3.4** For  $1 \leq i \leq \lfloor \frac{n}{2} \rfloor$ ,  $\binom{n}{i} > \binom{n}{i-1}$ .

**Proof.** This is by induction; the base case is apparent from the first few rows. Write

$$\binom{n}{i} = \binom{n-1}{i-1} + \binom{n-1}{i}$$

$$\binom{n}{i-1} = \binom{n-1}{i-2} + \binom{n-1}{i-1}$$

Provided that  $1 \leq i \leq \lfloor \frac{n-1}{2} \rfloor$ , we know by the induction hypothesis that

$$\binom{n-1}{i} > \binom{n-1}{i-1}.$$

Provided that  $1 \leq i-1 \leq \lfloor \frac{n-1}{2} \rfloor$ , we know that

$$\binom{n-1}{i-1} > \binom{n-1}{i-2}.$$

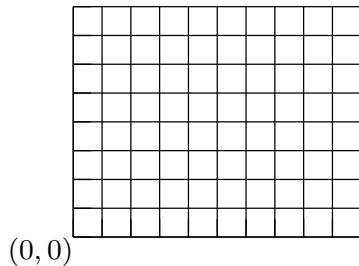
Hence if  $2 \leq i \leq \lfloor \frac{n-1}{2} \rfloor$ ,

$$\binom{n}{i} > \binom{n}{i-1}.$$

This leaves two special cases to check:  $i = 1$  and  $i \geq \lfloor \frac{n-1}{2} \rfloor + 1$ . These are left as an exercise. ■

### Exercises 1.3.

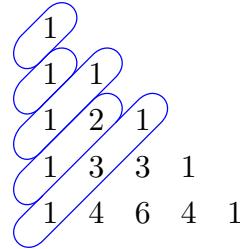
- Suppose a street grid starts at position  $(0, 0)$  and extends up and to the right:



A shortest route along streets from  $(0, 0)$  to  $(i, j)$  is  $i + j$  blocks long, going  $i$  blocks east and  $j$  blocks north. How many such routes are there? Suppose that the block between  $(k, l)$  and  $(k+1, l)$  is closed, where  $k < i$  and  $l \leq j$ . How many shortest routes are there from  $(0, 0)$  to  $(i, j)$ ?

- Prove by induction that  $\sum_{k=0}^n \binom{k}{i} = \binom{n+1}{i+1}$  for  $n \geq 0$  and  $i \geq 0$ .
- Use a combinatorial argument to prove that  $\sum_{k=0}^n \binom{k}{i} = \binom{n+1}{i+1}$  for  $n \geq 0$  and  $i \geq 0$ ; that is, explain why the left-hand side counts the same thing as the right-hand side.
- Use a combinatorial argument to prove that  $\binom{k}{2} + \binom{n-k}{2} + k(n-k) = \binom{n}{2}$ .
- Use a combinatorial argument to prove that  $\binom{2n}{n}$  is even.
- Suppose that  $A$  is a non-empty finite set. Prove that  $A$  has as many even-sized subsets as it does odd-sized subsets.

7. Prove that  $\sum_{k=0}^n \binom{k}{i} k = \binom{n+1}{i+1} n - \binom{n+1}{i+2}$  for  $n \geq 0$  and  $i \geq 0$ .
8. Verify that  $\binom{n+1}{2} + \binom{n}{2} = n^2$ . Use exercise 2 to find a simple expression for  $\sum_{i=1}^n i^2$ .
9. Make a conjecture about the sums of the upward diagonals in Pascal's Triangle as indicated. Prove your conjecture is true.



10. Find the number of ways to write  $n$  as an ordered sum of ones and twos,  $n \geq 0$ . For example, when  $n = 4$ , there are five ways:  $1 + 1 + 1 + 1$ ,  $2 + 1 + 1$ ,  $1 + 2 + 1$ ,  $1 + 1 + 2$ , and  $2 + 2$ .
11. Use  $(x+1)^n = \sum_{i=0}^n \binom{n}{i} x^i$  to find a simple expression for  $\sum_{i=0}^n \frac{1}{i+1} \binom{n}{i} x^{i+1}$ . Then find a simple expression for  $\sum_{i=0}^n \frac{1}{i+1} \binom{n}{i}$ .
12. Use the previous exercise to find a simple expression for  $\sum_{i=0}^n (-1)^i \frac{1}{i+1} \binom{n}{i}$ .
13. Give a combinatorial proof of

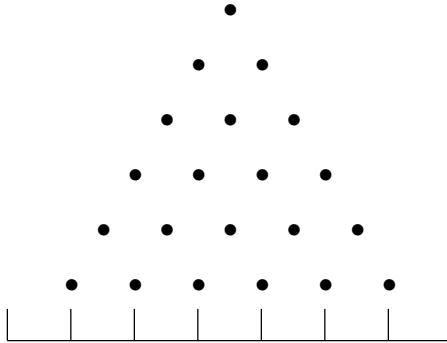
$$\sum_{i=0}^k \binom{m}{i} \binom{n}{k-i} = \binom{m+n}{k}.$$

Rewrite this identity in simpler form if  $m = n$ , and when  $k = m = n$ .

14. Finish the proof of theorem 1.3.4.
15. Give an alternate proof of theorem 1.3.4 by characterizing those  $i$  for which  $\binom{n}{i}/\binom{n}{i-1} > 1$ .
16. When is  $\binom{n}{i}/\binom{n}{i-1}$  a maximum? When is  $\binom{n}{i}/\binom{n}{i-1} = 2$ ?
17. When is  $\binom{n}{i} - \binom{n}{i-1}$  a maximum?
18. A **Galton board** is an upright flat surface with protruding horizontal pins arranged as shown below. At the bottom are a number of bins; if the number of rows is  $n$ , the number of bins is  $n+1$ . A ball is dropped directly above the top pin, and at each pin bounces left or right with equal probability. We assume that the ball next hits the pin below and immediately left or right of the pin it has struck, and this continues down the board, until the ball falls into a bin at the bottom. If we number the bins from 0 to  $n$ , how many paths can a ball travel to end up in bin  $k$ ?

This may be interpreted in terms of probability, which was the intent of Sir Francis Galton when he designed it. Each path is equally likely to be taken by a ball. If many balls are dropped, the number of balls in bin  $k$  corresponds to the probability of ending up in that bin. The more paths that end in a bin, the higher the probability. When a very large number of balls are dropped, the balls will form an approximation to the bell curve familiar from probability and statistics. There is an animation of the process at <http://www.math.uah.edu/stat/apps/GaltonBoardExperiment.html>. There was once a

very nice physical implementation at the [Pacific Science Center](#) in Seattle.



## 1.4 BELL NUMBERS

We begin with a definition:

**DEFINITION 1.4.1** A **partition** of a set  $S$  is a collection of non-empty subsets  $A_i \subseteq S$ ,  $1 \leq i \leq k$  (the **parts** of the partition), such that  $\bigcup_{i=1}^k A_i = S$  and for every  $i \neq j$ ,  $A_i \cap A_j = \emptyset$ .  $\square$

**EXAMPLE 1.4.2** The partitions of the set  $\{a, b, c\}$  are  $\{\{a\}, \{b\}, \{c\}\}$ ,  $\{\{a, b\}, \{c\}\}$ ,  $\{\{a, c\}, \{b\}\}$ ,  $\{\{b, c\}, \{a\}\}$ , and  $\{\{a, b, c\}\}$ .  $\square$

Partitions arise in a number of areas of mathematics. For example, if  $\equiv$  is an equivalence relation on a set  $S$ , the equivalence classes of  $\equiv$  form a partition of  $S$ . Here we consider the number of partitions of a finite set  $S$ , which we might as well take to be  $[n] = \{1, 2, 3, \dots, n\}$ , unless some other set is of interest. We denote the number of partitions of an  $n$ -element set by  $B_n$ ; these are the **Bell numbers**. From the example above, we see that  $B_3 = 5$ . For convenience we let  $B_0 = 1$ . It is quite easy to see that  $B_1 = 1$  and  $B_2 = 2$ .

There are no known simple formulas for  $B_n$ , so we content ourselves with a recurrence relation.

**THEOREM 1.4.3** The Bell numbers satisfy

$$B_{n+1} = \sum_{k=0}^n \binom{n}{k} B_k.$$

**Proof.** Consider a partition of  $S = \{1, 2, \dots, n+1\}$ ,  $A_1, \dots, A_m$ . We may suppose that  $n+1$  is in  $A_1$ , and that  $|A_1| = k+1$ , for some  $k$ ,  $0 \leq k \leq n$ . Then  $A_2, \dots, A_m$  form a partition of the remaining  $n-k$  elements of  $S$ , that is, of  $S \setminus A_1$ . There are  $B_{n-k}$  partitions

of this set, so there are  $B_{n-k}$  partitions of  $S$  in which one part is the set  $A_1$ . There are  $\binom{n}{k}$  sets of size  $k+1$  containing  $n+1$ , so the total number of partitions of  $S$  in which  $n+1$  is in a set of size  $k+1$  is  $\binom{n}{k}B_{n-k}$ . Adding up over all possible values of  $k$ , this means

$$B_{n+1} = \sum_{k=0}^n \binom{n}{k} B_{n-k}. \quad (1.4.1)$$

We may rewrite this, using theorem 1.3.3, as

$$B_{n+1} = \sum_{k=0}^n \binom{n}{n-k} B_{n-k},$$

and then notice that this is the same as the sum

$$B_{n+1} = \sum_{k=0}^n \binom{n}{k} B_k,$$

written backwards. ■

**EXAMPLE 1.4.4** We apply the recurrence to compute the first few Bell numbers:

$$\begin{aligned} B_1 &= \sum_{k=0}^0 \binom{0}{0} B_0 = 1 \cdot 1 = 1 \\ B_2 &= \sum_{k=0}^1 \binom{1}{k} B_k = \binom{1}{0} B_0 + \binom{1}{1} B_1 = 1 \cdot 1 + 1 \cdot 1 = 1 + 1 = 2 \\ B_3 &= \sum_{k=0}^2 \binom{2}{k} B_k = 1 \cdot 1 + 2 \cdot 1 + 1 \cdot 2 = 5 \\ B_4 &= \sum_{k=0}^3 \binom{3}{k} B_k = 1 \cdot 1 + 3 \cdot 1 + 3 \cdot 2 + 1 \cdot 5 = 15 \end{aligned}$$

□

The Bell numbers grow exponentially fast; the first few are 1, 1, 2, 5, 15, 52, 203, 877, 4140, 21147, 115975, 678570, 4213597, 27644437.

The Bell numbers turn up in many other problems; here is an interesting example. A common need in some computer programs is to generate a random permutation of  $1, 2, 3, \dots, n$ , which we may think of as a shuffle of the numbers, visualized as numbered cards in a deck. Here is an attractive method that is easy to program: Start with the numbers in order, then at each step, remove one number at random (this is easy in most programming languages) and put it at the front of the list of numbers. (Viewed as a shuffle

of a deck of cards, this corresponds to removing a card and putting it on the top of the deck.) How many times should we do this? There is no magic number, but it certainly should not be small relative to the size of  $n$ . Let's choose  $n$  as the number of steps.

We can write such a shuffle as a list of  $n$  integers,  $(m_1, m_2, \dots, m_n)$ . This indicates that at step  $i$ , the number  $m_i$  is moved to the front.

**EXAMPLE 1.4.5** Let's follow the shuffle  $(3, 2, 2, 4, 1)$ :

$$\begin{aligned} (3) : & 31245 \\ (2) : & 23145 \\ (2) : & 23145 \\ (4) : & 42315 \\ (1) : & 14235 \end{aligned}$$

□

Note that we allow “do nothing” moves, removing the top card and then placing it on top. The number of possible shuffles is then easy to count: there are  $n$  choices for the card to remove, and this is repeated  $n$  times, so the total number is  $n^n$ . (If we continue a shuffle for  $m$  steps, the number of shuffles is  $n^m$ .) Since there are only  $n!$  different permutations of  $1, 2, \dots, n$ , this means that many shuffles give the same final order.

Here's our question: how many shuffles result in the original order?

**EXAMPLE 1.4.6** These shuffles return to the original order:  $(1, 1, 1, 1, 1)$ ,  $(5, 4, 3, 2, 1)$ ,  $(4, 1, 3, 2, 1)$ . □

**THEOREM 1.4.7** The number of shuffles of  $[n]$  that result in the original sorted order is  $B_n$ .

**Proof.** Since we know that  $B_n$  counts the number of partitions of  $\{1, 2, 3, \dots, n\}$ , we can prove the theorem by establishing a 1–1 correspondence between the shuffles that leave the deck sorted and the partitions. Given a shuffle  $(m_1, m_2, \dots, m_n)$ , we put into a single set all  $i$  such that  $m_i$  has a single value. For example, using the shuffle  $(4, 1, 3, 2, 1)$ , Since  $m_2 = m_5$ , one set is  $\{2, 5\}$ . All the other values are distinct, so the other sets in the partition are  $\{1\}$ ,  $\{3\}$ , and  $\{4\}$ .

Note that every shuffle, no matter what the final order, produces a partition by this method. We are only interested in the shuffles that leave the deck sorted. What we now need is to show that each partition results from exactly one such shuffle.

Suppose we have a partition with  $k$  parts. If a shuffle leaves the deck sorted, the last entry,  $m_n$ , must be 1. If the part containing  $n$  is  $A_1$ , then it must be that  $m_i = 1$  if and

only if  $i \in A_1$ . If  $k = 1$ , then the only part contains all of  $\{1, 2, \dots, n\}$ , and the shuffle must be  $(1, 1, 1, \dots, 1)$ .

If  $k > 1$ , the last move that is not 1 must be 2, since 2 must end up immediately after 1. Thus, if  $j_2$  is the largest index such that  $j_2 \notin A_1$ , let  $A_2$  be the part containing  $j_2$ , and it must be that  $m_i = 2$  if and only if  $i \in A_2$ . We continue in this way: Once we have discovered which of the  $m_i$  must have values  $1, 2, \dots, p$ , let  $j_{p+1}$  be the largest index such that  $j_{p+1} \notin A_1 \cup \dots \cup A_p$ , let  $A_{p+1}$  be the part containing  $j_{p+1}$ , and then  $m_i = p + 1$  if and only if  $i \in A_{p+1}$ . When  $p = k$ , all values  $m_i$  have been determined, and this is the unique shuffle that corresponds to the partition. ■

**EXAMPLE 1.4.8** Consider the partition  $\{\{1, 5\}, \{2, 3, 6\}, \{4, 7\}\}$ . We must have  $m_7 = m_4 = 1$ ,  $m_6 = m_3 = m_2 = 2$ , and  $m_5 = m_1 = 3$ , so the shuffle is  $(3, 2, 2, 1, 3, 2, 1)$ . □

Returning to the problem of writing a computer program to generate a partition: is this a good method? When we say we want a random permutation, we mean that we want each permutation to occur with equal probability, namely,  $1/n!$ . Since the original order is one of the permutations, we want the number of shuffles that produce it to be exactly  $n^n/n!$ , but  $n!$  does not divide  $n^n$ , so this is impossible. The probability of getting the original permutation is  $B_n/n^n$ , and this turns out to be quite a bit larger than  $1/n!$ . Thus, this is not a suitable method for generating random permutations.

The recurrence relation above is a somewhat cumbersome way to compute the Bell numbers. Another way to compute them is with a different recurrence, expressed in the Bell triangle, whose construction is similar to that of Pascal's triangle:

$$\begin{array}{ccccccccc} A_{1,1} & & & & & & & & 1 \\ A_{2,1} & A_{2,2} & & & & & & & 1 \quad 2 \\ A_{3,1} & A_{3,2} & A_{3,3} & & & & & & 2 \quad 3 \quad 5 \\ A_{4,1} & A_{4,2} & A_{4,3} & A_{4,4} & & & & & 5 \quad 7 \quad 10 \quad 15 \end{array}$$

The rule for constructing this triangle is:  $A_{1,1} = 1$ ; the first entry in each row is the last entry in the previous row; other entries are  $A_{n,k} = A_{n,k-1} + A_{n-1,k-1}$ ; row  $n$  has  $n$  entries. Both the first column and the diagonal consist of the Bell numbers, with  $A_{n,1} = B_{n-1}$  and  $A_{n,n} = B_n$ .

$A_{n,k}$  may be interpreted as the number of partitions of  $\{1, 2, \dots, n+1\}$  in which  $\{k+1\}$  is the singleton set with the largest entry in the partition. For example,  $A_{3,2} = 3$ ; the partitions of  $3+1=4$  in which  $2+1=3$  is the largest number appearing in a singleton set are  $\{\{1\}, \{2, 4\}, \{3\}\}$ ,  $\{\{2\}, \{1, 4\}, \{3\}\}$ , and  $\{\{1, 2, 4\}, \{3\}\}$ .

To see that this indeed works as advertised, we need to confirm a few things. First, consider  $A_{n,n}$ , the number of partitions of  $\{1, 2, \dots, n+1\}$  in which  $\{n+1\}$  is the singleton set with the largest entry in the partition. Since  $n+1$  is the largest element of the set, all partitions containing the singleton  $\{n+1\}$  satisfy the requirement, and so the  $B_n$

## 26 Chapter 1 Fundamentals

partitions of  $\{1, 2, \dots, n\}$  together with  $\{n+1\}$  are exactly the partitions of interest, that is,  $A_{n,n} = B_n$ .

Next, we verify that under the desired interpretation, it is indeed true that  $A_{n,k} = A_{n,k-1} + A_{n-1,k-1}$  for  $k > 1$ . Consider a partition counted by  $A_{n,k-1}$ . This contains the singleton  $\{k\}$ , and the element  $k+1$  is not in a singleton. If we interchange  $k$  and  $k+1$ , we get the singleton  $\{k+1\}$ , and no larger element is in a singleton. This gives us partitions in which  $\{k+1\}$  is a singleton and  $\{k\}$  is not. Now consider a partition of  $\{1, 2, \dots, n\}$  counted by  $A_{n-1,k-1}$ . Replace all numbers  $j > k$  by  $j+1$ , and add the singleton  $\{k+1\}$ . This produces a partition in which both  $\{k+1\}$  and  $\{k\}$  appear. In fact, we have described how to produce each partition counted by  $A_{n,k}$  exactly once, and so  $A_{n,k} = A_{n,k-1} + A_{n-1,k-1}$ .

Finally, we need to verify that  $A_{n,1} = B_{n-1}$ . We know that  $A_{1,1} = 1 = B_0$ . Now we claim that for  $n > 1$ ,

$$A_{n,1} = \sum_{k=0}^{n-2} \binom{n-2}{k} A_{k+1,1}.$$

In a partition counted by  $A_{n,1}$ , 2 is the largest element in a singleton, so  $\{n+1\}$  is not in the partition. Choose any  $k \geq 1$  elements of  $\{3, 4, \dots, n\}$  to form the set containing  $n+1$ . There are  $A_{n-k-1,1}$  partitions of the remaining  $n-k$  elements in which 2 is the largest element in a singleton. This accounts for  $\binom{n-2}{k} A_{n-k-1,1}$  partitions of  $\{1, 2, \dots, n+1\}$ , or over all  $k$ :

$$\sum_{k=1}^{n-2} \binom{n-2}{k} A_{n-k-1,1} = \sum_{k=1}^{n-2} \binom{n-2}{n-k-2} A_{n-k-1,1} = \sum_{k=0}^{n-3} \binom{n-2}{k} A_{k+1,1}.$$

We are missing those partitions in which 1 is in the part containing  $n+1$ . We may produce all such partitions by starting with a partition counted by  $A_{n-1,1}$  and adding  $n+1$  to the part containing 1. Now we have

$$A_{n,1} = A_{n-1,1} + \sum_{k=0}^{n-3} \binom{n-2}{k} A_{k+1,1} = \sum_{k=0}^{n-2} \binom{n-2}{k} A_{k+1,1}.$$

Although slightly disguised by the shifted indexing of the  $A_{n,1}$ , this is the same as the recurrence relation for the  $B_n$ , and so  $A_{n,1} = B_{n-1}$  as desired.

### **Exercises 1.4.**

1. Show that if  $\{A_1, A_2, \dots, A_k\}$  is a partition of  $\{1, 2, \dots, n\}$ , then there is a unique equivalence relation  $\equiv$  whose equivalence classes are  $\{A_1, A_2, \dots, A_k\}$ .
2. Suppose  $n$  is a square-free number, that is, no number  $m^2$  divides  $n$ ; put another way, square-free numbers are products of distinct prime factors, that is,  $n = p_1 p_2 \cdots p_k$ , where each  $p_i$

is prime and no two prime factors are equal. Find the number of factorizations of  $n$ . For example,  $30 = 2 \cdot 3 \cdot 5$ , and the factorizations of 30 are 30,  $6 \cdot 5$ ,  $10 \cdot 3$ ,  $2 \cdot 15$ , and  $2 \cdot 3 \cdot 5$ . Note we count 30 alone as a factorization, though in some sense a trivial factorization.

3. The rhyme scheme of a stanza of poetry indicates which lines rhyme. This is usually expressed in the form ABAB, meaning the first and third lines of a four line stanza rhyme, as do the second and fourth, or ABCB, meaning only lines two and four rhyme, and so on. A limerick is a five line poem with rhyming scheme AABBA. How many different rhyme schemes are possible for an  $n$  line stanza? To avoid duplicate patterns, we only allow a new letter into the pattern when all previous letters have been used to the left of the new one. For example, ACBA is not allowed, since when C is placed in position 2, B has not been used to the left. This is the same rhyme scheme as ABCA, which is allowed.
4. Another way to express the Bell numbers for  $n > 0$  is

$$B_n = \sum_{k=1}^n S(n, k),$$

where  $S(n, k)$  is the number of partitions of  $\{1, 2, \dots, n\}$  into exactly  $k$  parts,  $1 \leq k \leq n$ . The  $S(n, k)$  are the **Stirling numbers of the second kind**. Find a recurrence relation for  $S(n, k)$ . Your recurrence should allow a fairly simple triangle construction containing the values  $S(n, k)$ , and then the Bell numbers may be computed by summing the rows of this triangle. Show the first five rows of the triangle,  $n \in \{1, 2, \dots, 5\}$ .

5. Let  $A_n$  be the number of partitions of  $\{1, 2, \dots, n+1\}$  in which no consecutive integers are in the same part of the partition. For example, when  $n = 3$  these partitions are  $\{\{1\}, \{2\}, \{3\}, \{4\}\}$ ,  $\{\{1\}, \{2, 4\}, \{3\}\}$ ,  $\{\{1, 3\}, \{2\}, \{4\}\}$ ,  $\{\{1, 3\}, \{2, 4\}\}$ ,  $\{\{1, 4\}, \{2\}, \{3\}\}$ , so  $A_3 = 5$ . Let  $A(n, k)$  be the number of partitions of  $\{1, 2, \dots, n+1\}$  into exactly  $k$  parts, in which no consecutive integers are in the same part of the partition. Thus

$$A_n = \sum_{k=2}^{n+1} A(n, k).$$

Find a recurrence for  $A(n, k)$  and then show that  $A_n = B_n$ .

## 1.5 CHOICE WITH REPETITION

Most of the permutation and combination problems we have seen count choices made without repetition, as when we asked how many rolls of three dice are there in which each die has a different value. The exception was the simplest problem, asking for the total number of outcomes when two or three dice are rolled, a simple application of the multiplication principle. Typical permutation and combination problems can be interpreted in terms of drawing balls from a box, and implicitly or explicitly the rule is that a ball drawn from the box stays out of the box. If instead each ball is returned to the box after recording the draw, we get a problem essentially identical to the general dice problem. For example, if there are six balls, numbered 1–6, and we draw three balls with replacement, the number of possible outcomes is  $6^3$ . Another version of the problem does not replace the ball after each draw, but allows multiple “identical” balls to be in the box. For example, if a box

28 Chapter 1 Fundamentals

contains 18 balls numbered 1–6, three with each number, then the possible outcomes when three balls are drawn and not returned to the box is again  $6^3$ . If four balls are drawn, however, the problem becomes different.

Another, perhaps more mathematical, way to phrase such problems is to introduce the idea of a **multiset**. A multiset is like a set, except that elements may appear more than once. If  $\{a, b\}$  and  $\{b, c\}$  are ordinary sets, we say that the union  $\{a, b\} \cup \{b, c\}$  is  $\{a, b, c\}$ , not  $\{a, b, b, c\}$ . If we interpret these as multisets, however, we do write  $\{a, b, b, c\}$  and consider this to be different than  $\{a, b, c\}$ . To distinguish multisets from sets, and to shorten the expression in most cases, we use a **repetition number** with each element. For example, we will write  $\{a, b, b, c\}$  as  $\{1 \cdot a, 2 \cdot b, 1 \cdot c\}$ . By writing  $\{1 \cdot a, 1 \cdot b, 1 \cdot c\}$  we emphasize that this is a multiset, even though no element appears more than once. We also allow elements to be included an infinite number of times, indicated with  $\infty$  for the repetition number, like  $\{\infty \cdot a, 5 \cdot b, 3 \cdot c\}$ .

Generally speaking, problems in which repetition numbers are infinite are easier than those involving finite repetition numbers. Given a multiset  $A = \{\infty \cdot a_1, \infty \cdot a_2, \dots, \infty \cdot a_n\}$ , how many permutations of the elements of length  $k$  are there? That is, how many sequences  $x_1, x_2, \dots, x_k$  can be formed? This is easy: the answer is  $n^k$ .

Now consider combinations of a multiset, that is, submultisets: Given a multiset, how many submultisets of a given size does it have? We say that a multiset  $A$  is a submultiset of  $B$  if the repetition number of every element of  $A$  is less than or equal to its repetition number in  $B$ . For example,  $\{20 \cdot a, 5 \cdot b, 1 \cdot c\}$  is a submultiset of  $\{\infty \cdot a, 5 \cdot b, 3 \cdot c\}$ . A multiset is finite if it contains only a finite number of distinct elements, and the repetition numbers are all finite. Suppose again that  $A = \{\infty \cdot a_1, \infty \cdot a_2, \dots, \infty \cdot a_n\}$ ; how many finite submultisets does it have of size  $k$ ? This at first seems quite difficult, but put in the proper form it turns out to be a familiar problem. Imagine that we have  $k + n - 1$  “blank spaces”, like this:

.....

Now we place  $n - 1$  markers in some of these spots:

$\wedge$      $\wedge$      $\wedge$     ...     $\wedge$

This uniquely identifies a submultiset: fill all blanks up to the first  $\wedge$  with  $a_1$ , up to the second with  $a_2$ , and so on:

$$\wedge \ a_2 \ \wedge \ a_3 \ a_3 \ a_3 \ \wedge \ a_4 \ \dots \ a_{n-1} \ a_{n-1} \ \wedge \ a_n$$

So this pattern corresponds to the multiset  $\{1 \cdot a_2, 3 \cdot a_3, \dots, 1 \cdot a_n\}$ . Filling in the markers  $\wedge$  in all possible ways produces all possible submultisets of size  $k$ , so there are  $\binom{k+n-1}{n-1}$

such submultisets. Note that this is the same as  $\binom{k+n-1}{k}$ ; the hard part in practice is remembering that the  $-1$  goes with the  $n$ , not the  $k$ .

• • •

Summarizing the high points so far: The number of permutations of  $n$  things taken  $k$  at a time without replacement is  $P(n, k) = n!/(n - k)!$ ; the number of permutations of  $n$  things taken  $k$  at a time with replacement is  $n^k$ . The number of combinations of  $n$  things taken  $k$  at a time without replacement is  $\binom{n}{k}$ ; the number of combinations of  $n$  things taken  $k$  at a time with replacement is  $\binom{k+n-1}{k}$ .

• • •

If  $A = \{m_1 \cdot a_1, m_2 \cdot a_2, \dots, m_n \cdot a_n\}$ , similar questions can be quite hard. Here is an easier special case: How many permutations of the multiset  $A$  are there? That is, how many sequences consist of  $m_1$  copies of  $a_1$ ,  $m_1$  copies of  $a_2$ , and so on? This problem succumbs to overcounting: suppose to begin with that we can distinguish among the different copies of each  $a_i$ ; they might be colored differently for example: a red  $a_1$ , a blue  $a_1$ , and so on. Then we have an ordinary set with  $M = \sum_{i=1}^n m_i$  elements and  $M!$  permutations. Now if we ignore the colors, so that all copies of  $a_i$  look the same, we find that we have overcounted the desired permutations. Permutations with, say, the  $a_1$  items in the same positions all look the same once we ignore the colors of the  $a_1$ s. How many of the original permutations have this property?  $m_1!$  permutations will appear identical once we ignore the colors of the  $a_1$  items, since there are  $m_1!$  permutations of the colored  $a_1$ s in a given  $m_1$  positions. So after throwing out duplicates, the number of remaining permutations is  $M!/m_1!$  (assuming the other  $a_i$  are still distinguishable). Then the same argument applies to the  $a_2$ s: there are  $m_2!$  copies of each permutation once we ignore the colors of the  $a_2$ s, so there are  $\frac{M!}{m_1! m_2!}$  distinct permutations. Continuing in this way, we see that the number of distinct permutations once all colors are ignored is

$$\frac{M!}{m_1! m_2! \cdots m_n!}.$$

This is frequently written

$$\binom{M}{m_1 \ m_2 \ \dots \ m_n},$$

called a **multinomial coefficient**. Here the second row has  $n$  separate entries, not a single product entry. Note that if  $n = 2$  this is

$$\binom{M}{m_1 \ m_2} = \frac{M!}{m_1! m_2!} = \frac{M!}{m_1! (M - m_1)!} = \binom{M}{m_1}. \quad (1.5.1)$$

This is easy to see combinatorially: given  $\{m_1 \cdot a_1, m_2 \cdot a_2\}$  we can form a permutation by choosing the  $m_1$  places that will be occupied by  $a_1$ , filling in the remaining  $m_2$  places with

### 30 Chapter 1 Fundamentals

$a_2$ . The number of permutations is the number of ways to choose the  $m_1$  locations, which is  $\binom{M}{m_1}$ .

**EXAMPLE 1.5.1** How many solutions does  $x_1 + x_2 + x_3 + x_4 = 20$  have in non-negative integers? That is, how many 4-tuples  $(m_1, m_2, m_3, m_4)$  of non-negative integers are solutions to the equation? We have actually solved this problem: How many submultisets of size 20 are there of the multiset  $\{\infty \cdot a_1, \infty \cdot a_2, \infty \cdot a_3, \infty \cdot a_4\}$ ? A submultiset of size 20 is of the form  $\{m_1 \cdot a_1, m_2 \cdot a_2, m_3 \cdot a_3, m_4 \cdot a_4\}$  where  $\sum m_i = 20$ , and these are in 1–1 correspondence with the set of 4-tuples  $(m_1, m_2, m_3, m_4)$  of non-negative integers such that  $\sum m_i = 20$ . Thus, the number of solutions is  $\binom{20+4-1}{20}$ . This reasoning applies in general: the number of solutions to

$$\sum_{i=1}^n x_i = k$$

is

$$\binom{k+n-1}{k}.$$

□

This immediately suggests some generalizations: instead of the total number of solutions, we might want the number of solutions with the variables  $x_i$  in certain ranges, that is, we might require that  $m_i \leq x_i \leq M_i$  for some lower and upper bounds  $m_i$  and  $M_i$ .

Finite upper bounds can be difficult to deal with; if we require that  $0 \leq x_i \leq M_i$ , this is the same as counting the submultisets of  $\{M_1 \cdot a_1, M_2 \cdot a_2, \dots, M_n \cdot a_n\}$ . Lower bounds are easier to deal with.

**EXAMPLE 1.5.2** Find the number of solutions to  $x_1 + x_2 + x_3 + x_4 = 20$  with  $x_1 \geq 0$ ,  $x_2 \geq 1$ ,  $x_3 \geq 2$ ,  $x_4 \geq -1$ .

We can transform this to the initial problem in which all lower bounds are 0. The solutions we seek to count are the solutions of this altered equation:

$$x_1 + (x_2 - 1) + (x_3 - 2) + (x_4 + 1) = 18.$$

If we set  $y_1 = x_1$ ,  $y_2 = x_2 - 1$ ,  $y_3 = x_3 - 2$ , and  $y_4 = x_4 + 1$ , then  $(x_1, x_2, x_3, x_4)$  is a solution to this equation if and only if  $(y_1, y_2, y_3, y_4)$  is a solution to

$$y_1 + y_2 + y_3 + y_4 = 18,$$

and moreover the bounds on the  $x_i$  are satisfied if and only if  $y_i \geq 0$ . Since the number of solutions to the last equation is  $\binom{18+4-1}{18}$ , this is also the number of solutions to the original equation. □

**Exercises 1.5.**

1. Suppose a box contains 18 balls numbered 1–6, three balls with each number. When 4 balls are drawn without replacement, how many outcomes are possible? Do this in two ways: assuming that the order in which the balls are drawn matters, and then assuming that order does not matter.
2. How many permutations are there of the letters in Mississippi?
3. How many permutations are there of the multiset  $\{1 \cdot a_1, 1 \cdot a_2, \dots, 1 \cdot a_n\}$ ?
4. Let  $M = \sum_{i=1}^n m_i$ . If  $k_i < 0$ , let's say

$$\binom{M}{k_1 \ k_2 \ \dots \ k_n} = 0.$$

Prove that

$$\binom{M}{m_1 \ m_2 \ \dots \ m_n} = \sum_{i=1}^n \binom{M-1}{m_1 \ m_2 \ \dots \ (m_i-1) \ \dots \ m_n}.$$

Note that when  $n = 2$  this becomes

$$\binom{M}{m_1 \ m_2} = \binom{M-1}{(m_1-1) \ m_2} + \binom{M-1}{m_1 \ (m_2-1)}.$$

As noted above in equation 1.5.1, when  $n = 2$  we are really seeing ordinary binomial coefficients, and this can be rewritten as

$$\binom{M}{m_1} = \binom{M-1}{m_1-1} + \binom{M-1}{m_1},$$

which of course we already know.

5. The Binomial Theorem (1.3.1) can be written

$$(x+y)^n = \sum_{i+j=n} \binom{n}{i \ j} x^i y^j,$$

where the sum is over all non-negative integers  $i$  and  $j$  that sum to  $n$ . Prove that for  $m \geq 2$ ,

$$(x_1 + x_2 + \dots + x_m)^n = \sum \binom{n}{i_1 \ i_2 \ \dots \ i_m} x_1^{i_1} x_2^{i_2} \dots x_m^{i_m}.$$

where the sum is over all  $i_1, \dots, i_m$  such that  $i_1 + \dots + i_m = n$ .

6. Find the number of integer solutions to

$$x_1 + x_2 + x_3 + x_4 + x_5 = 50, x_1 \geq -3, x_2 \geq 0, x_3 \geq 4, x_4 \geq 2, x_5 \geq 12.$$

7. You and your spouse each take two gummy vitamins every day. You share a single bottle of 60 vitamins, 30 of one flavor and 30 of another. You each prefer a different flavor, but it seems childish to fish out two of each type (but not to take gummy vitamins). So you just take the first four that fall out and then divide them up according to your preferences. For example, if there are two of each flavor, you and your spouse get the vitamins you prefer, but if three of your preferred flavor come out, you get two of the ones you like and your spouse gets one of each. Of course, you start a new bottle every 15 days. On average, over a 15 day period, how many of the vitamins you take are the flavor you prefer? (From [fivethirtyeight.com](http://fivethirtyeight.com).)

## 1.6 THE PIGEONHOLE PRINCIPLE

A key step in many proofs consists of showing that two possibly different values are in fact the same. The **Pigeonhole principle** can sometimes help with this.

**THEOREM 1.6.1 Pigeonhole Principle** Suppose that  $n+1$  (or more) objects are put into  $n$  boxes. Then some box contains at least two objects.

**Proof.** Suppose each box contains at most one object. Then the total number of objects is at most  $1 + 1 + \cdots + 1 = n$ , a contradiction. ■

This seemingly simple fact can be used in surprising ways. The key typically is to put objects into boxes according to some rule, so that when two objects end up in the same box it is because they have some desired relationship.

**EXAMPLE 1.6.2** Among any 13 people, at least two share a birth month.

Label 12 boxes with the names of the months. Put each person in the box labeled with his or her birth month. Some box will contain at least two people, who share a birth month. □

**EXAMPLE 1.6.3** Suppose 5 pairs of socks are in a drawer. Picking 6 socks guarantees that at least one pair is chosen.

Label the boxes by “the pairs” (e.g., the red pair, the blue pair, the argyle pair, . . .). Put the 6 socks into the boxes according to description. □

Some uses of the principle are not nearly so straightforward.

**EXAMPLE 1.6.4** Suppose  $a_1, \dots, a_n$  are integers. Then some “consecutive sum”  $a_k + a_{k+1} + a_{k+2} + \cdots + a_{k+m}$  is divisible by  $n$ .

Consider these  $n$  sums:

$$\begin{aligned}s_1 &= a_1 \\ s_2 &= a_1 + a_2 \\ s_3 &= a_1 + a_2 + a_3 \\ &\vdots \\ s_n &= a_1 + a_2 + \cdots + a_n\end{aligned}$$

These are all consecutive sums, so if one of them is divisible by  $n$  we are done. If not, dividing each by  $n$  leaves a non-zero remainder,  $r_1 = s_1 \bmod n$ ,  $r_2 = s_2 \bmod n$ , and so on. These remainders have values in  $\{1, 2, 3, \dots, n-1\}$ . Label  $n-1$  boxes with these  $n-1$  values; put each of the  $n$  sums into the box labeled with its remainder mod  $n$ . Two sums

end up in the same box, meaning that  $s_i \bmod n = s_j \bmod n$  for some  $j > i$ ; hence  $s_j - s_i$  is divisible by  $n$ , and  $s_j - s_i = a_{i+1} + a_{i+2} + \cdots + a_j$ , as desired.  $\square$

A similar argument provides a proof of the **Chinese Remainder Theorem**.

**THEOREM 1.6.5 Chinese Remainder Theorem** If  $m$  and  $n$  are relatively prime, and  $0 \leq a < m$  and  $0 \leq b < n$ , then there is an integer  $x$  such that  $x \bmod m = a$  and  $x \bmod n = b$ .

**Proof.** Consider the integers  $a, a+m, a+2m, \dots, a+(n-1)m$ , each with remainder  $a$  when divided by  $m$ . We wish to show that one of these integers has remainder  $b$  when divided by  $n$ , in which case that number satisfies the desired property.

For a contradiction, suppose not. Let the remainders be  $r_0 = a \bmod n, r_1 = a+m \bmod n, \dots, r_{n-1} = a+(n-1)m \bmod n$ . Label  $n-1$  boxes with the numbers  $0, 1, 2, 3, \dots, b-1, b+1, \dots, n-1$ . Put each  $r_i$  into the box labeled with its value. Two remainders end up in the same box, say  $r_i$  and  $r_j$ , with  $j > i$ , so  $r_i = r_j = r$ . This means that

$$a+im = q_1n+r \quad \text{and} \quad a+jm = q_2n+r.$$

Hence

$$\begin{aligned} a+jm - (a+im) &= q_2n+r - (q_1n+r) \\ (j-i)m &= (q_2-q_1)n. \end{aligned}$$

Since  $n$  is relatively prime to  $m$ , this means that  $n \mid (j-i)$ . But since  $i$  and  $j$  are distinct and in  $\{0, 1, 2, \dots, n-1\}$ ,  $0 < j-i < n$ , so  $n \nmid (j-i)$ . This contradiction finishes the proof.  $\blacksquare$

More general versions of the Pigeonhole Principle can be proved by essentially the same method. A natural generalization would be something like this: *If  $X$  objects are put into  $n$  boxes, some box contains at least  $m$  objects.* For example:

**THEOREM 1.6.6** Suppose that  $r_1, \dots, r_n$  are positive integers. If  $X \geq (\sum_{i=1}^n r_i) - n + 1$  objects are put into  $n$  boxes labeled  $1, 2, 3, \dots, n$ , then some box labeled  $i$  contains at least  $r_i$  objects.

**Proof.** Suppose not. Then the total number of objects in the boxes is at most  $(r_1-1) + (r_2-1) + (r_3-1) + \cdots + (r_n-1) = (\sum_{i=1}^n r_i) - n < X$ , a contradiction.  $\blacksquare$

This full generalization is only occasionally needed; often this simpler version is sufficient:

**COROLLARY 1.6.7** Suppose  $r > 0$  and  $X \geq n(r-1) + 1$  objects are placed into  $n$  boxes. Then some box contains at least  $r$  objects.

**Proof.** Apply the previous theorem with  $r_i = r$  for all  $i$ . ■

• • •

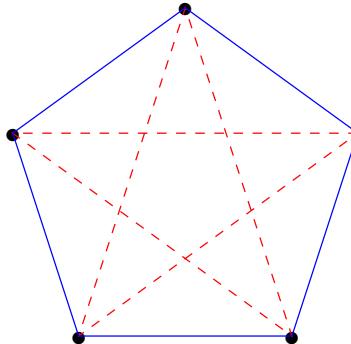
Here is a simple application of the Pigeonhole Principle that leads to many interesting questions.

**EXAMPLE 1.6.8** Suppose 6 people are gathered together; then either 3 of them are mutually acquainted, or 3 of them are mutually unacquainted.

We turn this into a graph theory question: Consider the graph consisting of 6 vertices, each connected to all the others by an edge, called the **complete graph** on 6 vertices, and denoted  $K_6$ ; the vertices represent the people. Color an edge red if the people represented by its endpoints are acquainted, and blue if they are not acquainted. Any choice of 3 vertices defines a triangle; we wish to show that either there is a red triangle or a blue triangle.

Consider the five edges incident at a single vertex  $v$ ; by the Pigeonhole Principle (the version in corollary 1.6.7, with  $r = 3$ ,  $X = 2(3 - 1) + 1 = 5$ ), at least three of them are the same color, call it color  $C$ ; call the other color  $D$ . Let the vertices at the other ends of these three edges be  $v_1, v_2, v_3$ . If any of the edges between these vertices have color  $C$ , there is a triangle of color  $C$ : if the edge connects  $v_i$  to  $v_j$ , the triangle is formed by  $v, v_i$ , and  $v_j$ . If this is not the case, then the three vertices  $v_1, v_2, v_3$  are joined by edges of color  $D$ , and form a triangle of color  $D$ . □

The number 6 in this example is special: with 5 or fewer vertices it is not true that there must be a monochromatic triangle, and with more than 6 vertices it is true. To see that it is not true for 5 vertices, we need only show an example, as in figure 1.6.1.



**Figure 1.6.1** An edge coloring with no monochromatic triangles.

The **Ramsey number**  $R(i)$  is the smallest integer  $n$  such that when the edges of  $K_n$  are colored with two colors, there is a monochromatic complete graph on  $i$  vertices,  $K_i$ , contained within  $K_n$ . The example shows that  $R(3) = 6$ .

More generally,  $R(i, j)$  is the smallest integer  $n$  such that when the edges of  $K_n$  are colored with two colors, say  $C_1$  and  $C_2$ , either there is a  $K_i$  contained within  $K_n$  all of whose edges are color  $C_1$ , or there is a  $K_j$  contained within  $K_n$  all of whose edges are color  $C_2$ . Using this notion,  $R(k) = R(k, k)$ . More generally still,  $R(i_1, i_2, \dots, i_m)$  is the smallest integer  $n$  such that when the edges of  $K_n$  are colored with  $m$  colors,  $C_1, \dots, C_m$ , then for some  $j$  there is a  $K_{i_j}$  contained in  $K_n$  all of whose edges are color  $C_j$ .

Ramsey proved that in all of these cases, there actually is such a number  $n$ . Generalizations of this problem have led to the subject called **Ramsey Theory**.

Computing any particular value  $R(i, j)$  turns out to be quite difficult; Ramsey numbers are known only for a few small values of  $i$  and  $j$ , and in some other cases the Ramsey number is bounded by known numbers. Typically in these cases someone has exhibited a  $K_m$  and a coloring of the edges without the existence of a monochromatic  $K_i$  or  $K_j$  of the desired color, showing that  $R(i, j) > m$ ; and someone has shown that whenever the edges of  $K_n$  have been colored, there is a  $K_i$  or  $K_j$  of the correct color, showing that  $R(i, j) \leq n$ .

### **Exercises 1.6.**

1. Assume that the relation “friend” is symmetric. Show that if  $n \geq 2$ , then in any group of  $n$  people there are two with the same number of friends in the group.
2. Suppose that 501 distinct integers are selected from  $1 \dots 1000$ . Show that there are distinct selected integers  $a$  and  $b$  such that  $a \mid b$ . Show that this is not always true if 500 integers are selected.  $\Rightarrow$
3. Each of 15 red balls and 15 green balls is marked with an integer between 1 and 100 inclusive; no integer appears on more than one ball. The value of a pair of balls is the sum of the numbers on the balls. Show there are at least two pairs, consisting of one red and one green ball, with the same value. Show that this is not necessarily true if there are 13 balls of each color.
4. Suppose we have 14 red balls and 14 green balls as in the previous exercise. Show that at least two pairs, consisting of one red and one green ball, have the same value. What about 13 red balls and 14 green balls?
5. Suppose  $(a_1, a_2, \dots, a_{52})$  are integers, not necessarily distinct. Show that there are two,  $a_i$  and  $a_j$  with  $i \neq j$ , such that either  $a_i + a_j$  or  $a_i - a_j$  is divisible by 100. Show that this is not necessarily true for integers  $(a_1, a_2, \dots, a_{51})$ .
6. Suppose five points are chosen from a square whose sides are length  $s$ . (The points may be either in the interior of the square or on the boundary.) Show that two of the points are at most  $s\sqrt{2}/2$  apart. Find five points so that no two are less than  $s\sqrt{2}/2$  apart.
7. Show that if the edges of  $K_6$  are colored with two colors, there are at least two monochromatic triangles. (Two triangles are different if each contains at least one vertex not in the other. For example, two red triangles that share an edge count as two triangles.) Color the edges of  $K_6$  so that there are exactly two monochromatic triangles.
8. Suppose the edges of a  $K_5$  are colored with two colors, say red and blue, so that there are no monochromatic triangles. Show that the red edges form a cycle, and the blue edges form

a cycle, each with five edges. (A **cycle** is a sequence of edges  $\{v_1, v_2\}, \{v_2, v_3\}, \dots, \{v_k, v_1\}$ , where all of the  $v_i$  are distinct. Note that this is true in figure 1.6.1.)

9. Show that  $8 < R(3, 4) \leq 10$ .
10. Show that  $R(3, 4) = 9$ .

## 1.7 SPERNER'S THEOREM

The binomial coefficients count the subsets of a given set; the sets themselves are worth looking at. First some convenient notation:

**DEFINITION 1.7.1** Let  $[n] = \{1, 2, 3, \dots, n\}$ . Then  $2^{[n]}$  denotes the set of all subsets of  $[n]$ , and  $\binom{[n]}{k}$  denotes the set of subsets of  $[n]$  of size  $k$ .  $\square$

**EXAMPLE 1.7.2** Let  $n = 3$ . Then

$$\begin{aligned}\binom{[n]}{0} &= \{\emptyset\} \\ \binom{[n]}{1} &= \{\{1\}, \{2\}, \{3\}\} \\ \binom{[n]}{2} &= \{\{1, 2\}, \{1, 3\}, \{2, 3\}\} \\ \binom{[n]}{3} &= \{\{1, 2, 3\}\}\end{aligned}$$

$\square$

**DEFINITION 1.7.3** A **chain** in  $2^{[n]}$  is a set of subsets of  $2^{[n]}$  that are linearly ordered by inclusion. An **anti-chain** in  $2^{[n]}$  is a set of subsets of  $2^{[n]}$  that are pairwise incomparable.  $\square$

**EXAMPLE 1.7.4** In  $2^{[3]}$ ,  $\{\emptyset, \{1\}, \{1, 2, 3\}\}$  is a chain, because  $\emptyset \subseteq \{1\} \subseteq \{1, 2, 3\}$ . Every  $\binom{[n]}{k}$  is an anti-chain, as is  $\{\{1\}, \{2, 3\}\}$ . The set  $\{\{1\}, \{1, 3\}, \{2, 3\}\}$  is neither a chain nor an anti-chain.  $\square$

Because of theorem 1.3.4 we know that among all anti-chains of the form  $\binom{[n]}{k}$  the largest are the “middle” ones, namely  $\binom{n}{\lfloor n/2 \rfloor}$  and  $\binom{n}{\lceil n/2 \rceil}$  (which are the same if  $n$  is even). Remarkably, these are the largest of all anti-chains, that is, strictly larger than every other anti-chain. When  $n = 3$ , the anti-chains  $\binom{[3]}{1}$  and  $\binom{[3]}{2}$  are the only anti-chains of size 3, and no anti-chain is larger, as you can verify by examining all possibilities.

Before we prove this, a bit of notation.

**DEFINITION 1.7.5** If  $\sigma: A \rightarrow A$  is a bijection, then  $\sigma$  is called a permutation.  $\square$

This use of the word permutation is different than our previous usage, but the two are closely related. Consider such a function  $\sigma: [n] \rightarrow [n]$ . Since the set  $A$  in this case is finite, we could in principle list every value of  $\sigma$ :

$$\sigma(1), \sigma(2), \sigma(3), \dots, \sigma(n).$$

This is a list of the numbers  $\{1, \dots, n\}$  in some order, namely, this is a permutation according to our previous usage. We can continue to use the same word for both ideas, relying on context or an explicit statement to indicate which we mean.

**THEOREM 1.7.6** (Sperner's Theorem) The only anti-chains of largest size are  $\binom{n}{\lfloor n/2 \rfloor}$  and  $\binom{n}{\lceil n/2 \rceil}$ .

**Proof.** First we show that no anti-chain is larger than these two. We attempt to partition  $2^{[n]}$  into  $k = \binom{n}{\lfloor n/2 \rfloor}$  chains, that is, to find chains

$$\begin{aligned} A_{1,0} &\subseteq A_{1,1} \subseteq A_{1,2} \subseteq \cdots \subseteq A_{1,m_1} \\ A_{2,0} &\subseteq A_{2,1} \subseteq A_{2,2} \subseteq \cdots \subseteq A_{2,m_2} \\ &\vdots \\ A_{k,0} &\subseteq A_{k,1} \subseteq A_{k,2} \subseteq \cdots \subseteq A_{k,m_k} \end{aligned}$$

so that every subset of  $[n]$  appears exactly once as one of the  $A_{i,j}$ . If we can find such a partition, then since no two elements of an anti-chain can be in the same chain, no anti-chain can have more than  $k$  elements.

For small values of  $n$  this can be done by hand; for  $n = 3$  we have

$$\begin{aligned} \emptyset &\subseteq \{1\} \subseteq \{1, 2\} \subseteq \{1, 2, 3\} \\ \{2\} &\subseteq \{2, 3\} \\ \{3\} &\subseteq \{1, 3\} \end{aligned}$$

These small cases form the base of an induction. We will prove that any  $2^{[n]}$  can be partitioned into such chains with two additional properties:

1. Each set in a chain contains exactly one element more than the next smallest set in the chain.
2. The sum of the sizes of the smallest and largest element in the chain is  $n$ .

Note that the chains for the case  $n = 3$  have both of these properties. The two properties taken together imply that every chain “crosses the middle”, that is, every chain contains an element of  $\binom{n}{n/2}$  if  $n$  is even, and an element of both  $\binom{n}{\lfloor n/2 \rfloor}$  and  $\binom{n}{\lceil n/2 \rceil}$  if  $n$  is odd.

### 38 Chapter 1 Fundamentals

Thus, if we succeed in showing that such chain partitions exist, there will be exactly  $\binom{n}{\lfloor n/2 \rfloor}$  chains.

For the induction step, we assume that we have partitioned  $2^{[n-1]}$  into such chains, and construct chains for  $2^{[n]}$ .

First, for each chain  $A_{i,0} \subseteq A_{i,1} \subseteq \dots \subseteq A_{i,m_i}$  we form a new chain  $A_{i,0} \subseteq A_{i,1} \subseteq \dots \subseteq A_{i,m_i} \subseteq A_{i,m_i} \cup \{n\}$ . Since  $|A_{i,0}| + |A_{i,m_i}| = n - 1$ ,  $|A_{i,0}| + |A_{i,m_i} \cup \{n\}| = n$ , so this new chain satisfies properties (1) and (2).

In addition, if  $m_i > 0$ , we form a new new chain  $A_{i,0} \cup \{n\} \subseteq A_{i,1} \cup \{n\} \subseteq \dots \subseteq A_{i,m_i-1} \cup \{n\}$ . Now

$$\begin{aligned} |A_{i,0} \cup \{n\}| + |A_{i,m_i-1} \cup \{n\}| &= |A_{i,0}| + 1 + |A_{i,m_i-1}| + 1 \\ &= |A_{i,0}| + 1 + |A_{i,m_i}| - 1 + 1 \\ &= n - 1 + 1 = n \end{aligned}$$

so again properties (1) and (2) are satisfied.

Because of the first type of chain, all subsets of  $[n-1]$  are contained exactly once in the new set of chains. Also, we have added the element  $n$  exactly once to every subset of  $[n-1]$ , so we have included every subset of  $[n]$  containing  $n$  exactly once. Thus we have produced the desired partition of  $2^{[n]}$ .

Now we need to show that the only largest anti-chains are  $\binom{n}{\lfloor n/2 \rfloor}$  and  $\binom{n}{\lceil n/2 \rceil}$ .

Suppose that  $A_1, A_2, \dots, A_m$  is an anti-chain; then  $A_1^c, A_2^c, \dots, A_m^c$  is also an anti-chain, where  $A^c$  denotes the complement of  $A$ . Thus, if there is an anti-chain that contains some  $A$  with  $|A| > \lceil n/2 \rceil$ , there is also one containing  $A^c$ , and  $|A^c| < \lfloor n/2 \rfloor$ . Suppose that some anti-chain contains a set  $A$  with  $|A| < \lfloor n/2 \rfloor$ . We next prove that this anti-chain cannot be of maximum size.

Partition  $2^{[n]}$  as in the first part of the proof. Suppose that  $A$  is a subset of the elements of a one or two element chain  $C$ , that is, a chain consisting solely of a set  $S_1$  of size  $n/2$ , if  $n$  is even, or of sets  $S_1$  and  $S_2$  of sizes  $\lfloor n/2 \rfloor$  and  $\lceil n/2 \rceil$ , with  $A \subseteq S_1 \subseteq S_2$ , if  $n$  is odd. Then no member of  $C$  is in the anti-chain. Thus, the largest possible size for an anti-chain containing  $A$  is  $\binom{n}{\lfloor n/2 \rfloor} - 1$ .

If  $A$  is not a subset of the elements of such a short chain, we now prove that there is another chain partition of  $2^{[n]}$  that does have this property. Note that in the original chain partition there must be a chain of length 1 or 2,  $C_1$ , consisting of  $S_1$  and possibly  $S_2$ ; if not, every chain would contain a set of size  $\lfloor n/2 \rfloor - 1$ , but there are not enough such sets to go around. Suppose then that  $A = \{x_1, \dots, x_k\}$  and the set  $S_1$  in  $C_1$  is  $S_1 = \{x_1, \dots, x_q, y_{q+1}, \dots, y_l\}$ , where  $0 \leq q < k$  and  $l > k$ .

Let  $\sigma$  be the permutation of  $[n]$  such that  $\sigma(x_{q+i}) = y_{q+i}$  and  $\sigma(y_{q+i}) = x_{q+i}$ , for  $1 \leq i \leq k - q$ , and  $\sigma$  fixes all other elements. Now for  $U \subseteq [n]$ , let  $\bar{U} = \sigma(U)$ , and note that  $U \subseteq V$  if and only if  $\bar{U} \subseteq \bar{V}$ . Thus every chain in the original chain partition maps

to a chain. Since  $\sigma$  is a bijection, these new chains also form a partition of  $2^{[n]}$ , with the additional properties (1) and (2). By the definition of  $\sigma$ ,  $A \subseteq \bar{S}_1$ , and  $\{\bar{S}_1, \bar{S}_2\}$  is a chain, say  $\bar{C}_1$ . Thus, this new chain partition has the desired property:  $A$  is a subset of every element of the 1 or 2 element chain  $\bar{C}_1$ , so  $A$  is not in an anti-chain of maximum size.

Finally, we need to show that if  $n$  is odd, no anti-chain of maximum size contains sets in both  $\binom{n}{\lfloor n/2 \rfloor}$  and  $\binom{n}{\lceil n/2 \rceil}$ . Suppose there is such an anti-chain, consisting of sets  $A_{k+1}, \dots, A_l$  in  $\binom{n}{\lceil n/2 \rceil}$ , where  $l = \binom{n}{\lceil n/2 \rceil}$ , and  $B_1, \dots, B_k$  in  $\binom{n}{\lfloor n/2 \rfloor}$ . The remaining sets in  $\binom{n}{\lceil n/2 \rceil}$  are  $A_1, \dots, A_k$ , and the remaining sets in  $\binom{n}{\lfloor n/2 \rfloor}$  are  $B_{k+1}, \dots, B_l$ .

Each set  $B_i$ ,  $1 \leq i \leq k$ , is contained in exactly  $\lceil n/2 \rceil$  sets in  $\binom{n}{\lceil n/2 \rceil}$ , and all must be among  $A_1, \dots, A_k$ . On average, then, each  $A_i$ ,  $1 \leq i \leq k$ , contains  $\lceil n/2 \rceil$  sets among  $B_1, \dots, B_k$ . But each set  $A_i$ ,  $1 \leq i \leq k$ , contains exactly  $\lceil n/2 \rceil$  sets in  $\binom{n}{\lfloor n/2 \rfloor}$ , and so each must contain exactly  $\lceil n/2 \rceil$  of the sets  $B_1, \dots, B_k$  and none of the sets  $B_{k+1}, \dots, B_l$ .

Let  $A_1 = A_{j_1} = \{x_1, \dots, x_r\}$  and  $B_{k+1} = \{x_1, \dots, x_s, y_{s+1}, \dots, y_{r-1}\}$ . Let  $B_{i_m} = A_{j_m} \setminus \{x_{s+m}\}$  and  $A_{j_{m+1}} = B_{i_m} \cup \{y_{s+m}\}$ , for  $1 \leq m \leq r-s-1$ . Note that by the preceding discussion,  $1 \leq i_m \leq k$  and  $1 \leq j_m \leq k$ . Then  $A_{j_{r-s}} = \{x_1, \dots, x_s, y_{s+1}, \dots, y_{r-1}, x_r\}$ , so  $A_{j_{r-s}} \supseteq B_{k+1}$ , a contradiction. Hence there is no such anti-chain. ■

### Exercises 1.7.

1. Sperner's Theorem (1.7.6) tells us that  $\binom{6}{3}$ , with size 20, is the unique largest anti-chain for  $2^{[6]}$ . The next largest anti-chains of the form  $\binom{6}{k}$  are  $\binom{6}{2}$  and  $\binom{6}{4}$ , with size 15. Find a maximal anti-chain with size larger than 15 but less than 20. (As usual, maximal here means that the anti-chain cannot be enlarged simply by adding elements. So you may not simply use a subset of  $\binom{6}{3}$ .)

## 1.8 STIRLING NUMBERS

In exercise 4 in section 1.4, we saw the Stirling numbers of the second kind. Not surprisingly, there are **Stirling numbers of the first kind**. Recall that Stirling numbers of the second kind are defined as follows:

**DEFINITION 1.8.1** The Stirling number of the second kind,  $S(n, k)$  or  $\binom{n}{k}$ , is the number of partitions of  $[n] = \{1, 2, \dots, n\}$  into exactly  $k$  parts,  $1 \leq k \leq n$ . □

Before we define the Stirling numbers of the first kind, we need to revisit permutations. As we mentioned in section 1.7, we may think of a permutation of  $[n]$  either as a reordering of  $[n]$  or as a bijection  $\sigma: [n] \rightarrow [n]$ . There are different ways to write permutations when thought of as functions. Two typical and useful ways are as a table, and in **cycle form**. Consider this permutation  $\sigma: [5] \rightarrow [5]$ :  $\sigma(1) = 3$ ,  $\sigma(2) = 4$ ,  $\sigma(3) = 5$ ,  $\sigma(4) = 2$ ,  $\sigma(5) = 1$ .

## 40 Chapter 1 Fundamentals

In table form, we write this as  $(\begin{smallmatrix} 1 & 2 & 3 & 4 & 5 \\ 3 & 4 & 5 & 2 & 1 \end{smallmatrix})$ , which is somewhat more compact, as we don't write " $\sigma$ " five times. In cycle form, we write this same permutation as  $(1, 3, 5)(2, 4)$ . Here  $(1, 3, 5)$  indicates that  $\sigma(1) = 3$ ,  $\sigma(3) = 5$ , and  $\sigma(5) = 1$ , while  $(2, 4)$  indicates  $\sigma(2) = 4$  and  $\sigma(4) = 2$ . This permutation has two cycles, a 3-cycle and a 2-cycle. Note that  $(1, 3, 5)$ ,  $(3, 5, 1)$ , and  $(5, 1, 3)$  all mean the same thing. We allow 1-cycles to count as cycles, though sometimes we don't write them explicitly. In some cases, however, it is valuable to write them to force us to remember that they are there. Consider this permutation:  $(\begin{smallmatrix} 1 & 2 & 3 & 4 & 5 & 6 \\ 3 & 4 & 5 & 2 & 1 & 6 \end{smallmatrix})$ . If we write this in cycle form as  $(1, 3, 5)(2, 4)$ , which is correct, there is no indication that the underlying set is really [6]. Writing  $(1, 3, 5)(2, 4)(6)$  makes this clear. We say that this permutation has 3 cycles, even though one of them is a trivial 1-cycle. Now we're ready for the next definition.

**DEFINITION 1.8.2** The **Stirling number of the first kind**,  $s(n, k)$ , is  $(-1)^{n-k}$  times the number of permutations of  $[n]$  with exactly  $k$  cycles. The corresponding **unsigned Stirling number of the first kind**, the number of permutations of  $[n]$  with exactly  $k$  cycles, is  $|s(n, k)|$ , sometimes written  $\begin{bmatrix} n \\ k \end{bmatrix}$ . Using this notation,  $s(n, k) = (-1)^{n-k} \begin{bmatrix} n \\ k \end{bmatrix}$ .  $\square$

Note that the use of  $\begin{bmatrix} n \\ k \end{bmatrix}$  conflicts with the use of the same notation in section 1.7; there should be no confusion, as we won't be discussing the two ideas together.

Some values of  $\begin{bmatrix} n \\ k \end{bmatrix}$  are easy to see; if  $n \geq 1$ , then

$$\begin{aligned} \begin{bmatrix} n \\ n \end{bmatrix} &= 1 & \begin{bmatrix} n \\ k \end{bmatrix} &= 0, \text{ if } k > n \\ \begin{bmatrix} n \\ 1 \end{bmatrix} &= (n-1)! & \begin{bmatrix} n \\ 0 \end{bmatrix} &= 0 \end{aligned}$$

It is sometimes convenient to say that  $\begin{bmatrix} 0 \\ 0 \end{bmatrix} = 1$ . These numbers thus form a triangle in the obvious way, just as the Stirling numbers of the first kind do. Here are lines 1–5 of the triangle:

$$\begin{array}{ccccccc} 1 & & & & & & \\ 0 & 1 & & & & & \\ 0 & 1 & 1 & & & & \\ 0 & 2 & 3 & 1 & & & \\ 0 & 6 & 11 & 6 & 1 & & \\ 0 & 24 & 50 & 35 & 10 & 1 & \end{array}$$

The first column is not particularly interesting, so often it is eliminated.

In exercise 4 in section 1.4, we saw that

$$\begin{Bmatrix} n \\ k \end{Bmatrix} = \begin{Bmatrix} n-1 \\ k-1 \end{Bmatrix} + k \cdot \begin{Bmatrix} n-1 \\ k \end{Bmatrix}. \quad (1.8.1)$$

The unsigned Stirling numbers of the first kind satisfy a similar recurrence.

**THEOREM 1.8.3**  $\left[ \begin{smallmatrix} n \\ k \end{smallmatrix} \right] = \left[ \begin{smallmatrix} n-1 \\ k-1 \end{smallmatrix} \right] + (n-1) \cdot \left[ \begin{smallmatrix} n-1 \\ k \end{smallmatrix} \right]$ ,  $k \geq 1, n \geq 1$ .

**Proof.** The proof is by induction on  $n$ ; the table above shows that it is true for the first few lines. We split the permutations of  $[n]$  with  $k$  cycles into two types: those in which  $(n)$  is a 1-cycle, and the rest. If  $(n)$  is a 1-cycle, then the remaining cycles form a permutation of  $[n-1]$  with  $k-1$  cycles, so there are  $\left[ \begin{smallmatrix} n-1 \\ k-1 \end{smallmatrix} \right]$  of these. Otherwise,  $n$  occurs in a cycle of length at least 2, and removing  $n$  leaves a permutation of  $[n-1]$  with  $k$  cycles. Given a permutation  $\sigma$  of  $[n-1]$  with  $k$  cycles,  $n$  can be added to any cycle in any position to form a permutation of  $[n]$  in which  $(n)$  is not a 1-cycle. Suppose the lengths of the cycles in  $\sigma$  are  $l_1, l_2, \dots, l_k$ . In cycle number  $i$ ,  $n$  may be added after any of the  $l_i$  elements in the cycle. Thus, the total number of places that  $n$  can be added is  $l_1 + l_2 + \dots + l_k = n-1$ , so there are  $(n-1) \cdot \left[ \begin{smallmatrix} n-1 \\ k \end{smallmatrix} \right]$  permutations of  $[n]$  in which  $(n)$  is not a 1-cycle. Now the total number of permutations of  $[n]$  with  $k$  cycles is  $\left[ \begin{smallmatrix} n-1 \\ k-1 \end{smallmatrix} \right] + (n-1) \cdot \left[ \begin{smallmatrix} n-1 \\ k \end{smallmatrix} \right]$ , as desired. ■

**COROLLARY 1.8.4**  $s(n, k) = s(n-1, k-1) - (n-1)s(n-1, k)$ . ■

The Stirling numbers satisfy two remarkable identities. First a definition:

**DEFINITION 1.8.5** The **Kronecker delta**  $\delta_{n,k}$  is 1 if  $n = k$  and 0 otherwise. □

**THEOREM 1.8.6** For  $n \geq 0$  and  $k \geq 0$ ,

$$\sum_{j=0}^n s(n, j)S(j, k) = \sum_{j=0}^n (-1)^{n-j} \left[ \begin{smallmatrix} n \\ j \end{smallmatrix} \right] \left\{ \begin{smallmatrix} j \\ k \end{smallmatrix} \right\} = \delta_{n,k}$$

$$\sum_{j=0}^n S(n, j)s(j, k) = \sum_{j=0}^n (-1)^{j-k} \left\{ \begin{smallmatrix} n \\ j \end{smallmatrix} \right\} \left[ \begin{smallmatrix} j \\ k \end{smallmatrix} \right] = \delta_{n,k}$$

**Proof.** We prove the first version, by induction on  $n$ . The first few values of  $n$  are easily checked; assume  $n > 1$ . Now note that  $\left[ \begin{smallmatrix} n \\ 0 \end{smallmatrix} \right] = 0$ , so we may start the sum index  $j$  at 1.

When  $k > n$ ,  $\left\{ \begin{smallmatrix} j \\ k \end{smallmatrix} \right\} = 0$ , for  $1 \leq j \leq n$ , and so the sum is 0. When  $k = n$ , the only non-zero term occurs when  $j = n$ , and is  $(-1)^0 \left[ \begin{smallmatrix} n \\ n \end{smallmatrix} \right] \left\{ \begin{smallmatrix} n \\ n \end{smallmatrix} \right\} = 1$ , so the sum is 1. Now suppose  $k < n$ . When  $k = 0$ ,  $\left\{ \begin{smallmatrix} j \\ k \end{smallmatrix} \right\} = 0$  for  $j > 0$ , so the sum is 0, and we assume now that  $k > 0$ .

## 42 Chapter 1 Fundamentals

We begin by applying the recurrence relations:

$$\begin{aligned}
\sum_{j=1}^n (-1)^{n-j} \begin{bmatrix} n \\ j \end{bmatrix} \begin{Bmatrix} j \\ k \end{Bmatrix} &= \sum_{j=1}^n (-1)^{n-j} \left( \begin{bmatrix} n-1 \\ j-1 \end{bmatrix} + (n-1) \begin{bmatrix} n-1 \\ j \end{bmatrix} \right) \begin{Bmatrix} j \\ k \end{Bmatrix} \\
&= \sum_{j=1}^n (-1)^{n-j} \begin{bmatrix} n-1 \\ j-1 \end{bmatrix} \begin{Bmatrix} j \\ k \end{Bmatrix} + \sum_{j=1}^n (-1)^{n-j} (n-1) \begin{bmatrix} n-1 \\ j \end{bmatrix} \begin{Bmatrix} j \\ k \end{Bmatrix} \\
&= \sum_{j=1}^n (-1)^{n-j} \begin{bmatrix} n-1 \\ j-1 \end{bmatrix} \left( \begin{Bmatrix} j-1 \\ k-1 \end{Bmatrix} + k \begin{Bmatrix} j-1 \\ k \end{Bmatrix} \right) + \sum_{j=1}^n (-1)^{n-j} (n-1) \begin{bmatrix} n-1 \\ j \end{bmatrix} \begin{Bmatrix} j \\ k \end{Bmatrix} \\
&= \sum_{j=1}^n (-1)^{n-j} \begin{bmatrix} n-1 \\ j-1 \end{bmatrix} \begin{Bmatrix} j-1 \\ k-1 \end{Bmatrix} + \sum_{j=1}^n (-1)^{n-j} \begin{bmatrix} n-1 \\ j-1 \end{bmatrix} k \begin{Bmatrix} j-1 \\ k \end{Bmatrix} \\
&\quad + \sum_{j=1}^n (-1)^{n-j} (n-1) \begin{bmatrix} n-1 \\ j \end{bmatrix} \begin{Bmatrix} j \\ k \end{Bmatrix}.
\end{aligned}$$

Consider the first sum in the last expression:

$$\begin{aligned}
\sum_{j=1}^n (-1)^{n-j} \begin{bmatrix} n-1 \\ j-1 \end{bmatrix} \begin{Bmatrix} j-1 \\ k-1 \end{Bmatrix} &= \sum_{j=2}^n (-1)^{n-j} \begin{bmatrix} n-1 \\ j-1 \end{bmatrix} \begin{Bmatrix} j-1 \\ k-1 \end{Bmatrix} \\
&= \sum_{j=1}^{n-1} (-1)^{n-j-1} \begin{bmatrix} n-1 \\ j \end{bmatrix} \begin{Bmatrix} j \\ k-1 \end{Bmatrix} \\
&= \delta_{n-1, k-1} = 0,
\end{aligned}$$

since  $k-1 < n-1$  (or trivially, if  $k=1$ ). Thus, we are left with just two sums.

$$\begin{aligned}
\sum_{j=1}^n (-1)^{n-j} \begin{bmatrix} n-1 \\ j-1 \end{bmatrix} k \begin{Bmatrix} j-1 \\ k \end{Bmatrix} &+ \sum_{j=1}^n (-1)^{n-j} (n-1) \begin{bmatrix} n-1 \\ j \end{bmatrix} \begin{Bmatrix} j \\ k \end{Bmatrix} \\
&= k \sum_{j=1}^{n-1} (-1)^{n-j-1} \begin{bmatrix} n-1 \\ j \end{bmatrix} \begin{Bmatrix} j \\ k \end{Bmatrix} - (n-1) \sum_{j=1}^{n-1} (-1)^{n-j-1} \begin{bmatrix} n-1 \\ j \end{bmatrix} \begin{Bmatrix} j \\ k \end{Bmatrix} \\
&= k \delta_{n-1, k} - (n-1) \delta_{n-1, k}.
\end{aligned}$$

Now if  $k=n-1$ , this is  $(n-1)\delta_{n-1, n-1} - (n-1)\delta_{n-1, n-1} = 0$ , while if  $k < n-1$  it is  $k\delta_{n-1, k} - (n-1)\delta_{n-1, k} = k \cdot 0 - (n-1) \cdot 0 = 0$ . ■

If we interpret the triangles containing the  $s(n, k)$  and  $S(n, k)$  as matrices, either  $m \times m$ , by taking the first  $m$  rows and columns, or even the infinite matrices containing the entire triangles, the sums of the theorem correspond to computing the matrix product in both orders. The theorem then says that this product consists of ones on the diagonal

and zeros elsewhere, so these matrices are inverses. Here is a small example:

$$\begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & -1 & 1 & 0 & 0 & 0 \\ 0 & 2 & -3 & 1 & 0 & 0 \\ 0 & -6 & 11 & -6 & 1 & 0 \\ 0 & 24 & -50 & 35 & -10 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 3 & 1 & 0 & 0 \\ 0 & 1 & 7 & 6 & 1 & 0 \\ 0 & 1 & 15 & 25 & 10 & 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

### Exercises 1.8.

1. Find a simple expression for  $\left[ \begin{smallmatrix} n \\ n-1 \end{smallmatrix} \right]$ .
2. Find a simple expression for  $\left[ \begin{smallmatrix} n \\ 1 \end{smallmatrix} \right]$ .
3. What is  $\sum_{k=0}^n \left\{ \begin{smallmatrix} n \\ k \end{smallmatrix} \right\}$ ?
4. What is  $\sum_{k=0}^n s(n, k)$ ?
5. Show that  $x^n = \prod_{k=0}^{n-1} (x - k) = \sum_{i=0}^n s(n, i)x^i$ ,  $n \geq 1$ ;  $x^n$  is called a **falling factorial**. Find a similar identity for  $x^{\bar{n}} = \prod_{k=0}^{n-1} (x + k)$ ;  $x^{\bar{n}}$  is a **rising factorial**.
6. Show that  $\sum_{k=0}^n \left\{ \begin{smallmatrix} n \\ k \end{smallmatrix} \right\} x^k = x^n$ ,  $n \geq 1$ ;  $x^k$  is defined in the previous exercise. The previous exercise shows how to express the falling factorial in terms of powers of  $x$ ; this exercise shows how to express the powers of  $x$  in terms of falling factorials.
7. Prove:  $S(n, k) = \sum_{i=k-1}^{n-1} \binom{n-1}{i} S(i, k-1)$ .
8. Prove:  $\left[ \begin{smallmatrix} n \\ k \end{smallmatrix} \right] = \sum_{i=k-1}^{n-1} (n-i-1)! \binom{n-1}{i} \left[ \begin{smallmatrix} i \\ k-1 \end{smallmatrix} \right]$ .
9. Use the previous exercise to prove  $s(n, k) = \sum_{i=k-1}^{n-1} (-1)^{n-i-1} (n-i-1)! \binom{n-1}{i} s(i, k-1)$ .
10. We have defined  $\left[ \begin{smallmatrix} n \\ k \end{smallmatrix} \right]$  and  $\left\{ \begin{smallmatrix} n \\ k \end{smallmatrix} \right\}$  for  $n, k \geq 0$ . We want to extend the definitions to all integers. Without some extra stipulations, there are many ways to do this. Let us suppose that for  $n \neq 0$  we want  $\left[ \begin{smallmatrix} n \\ 0 \end{smallmatrix} \right] = \left[ \begin{smallmatrix} 0 \\ n \end{smallmatrix} \right] = \left\{ \begin{smallmatrix} n \\ 0 \end{smallmatrix} \right\} = \left\{ \begin{smallmatrix} 0 \\ n \end{smallmatrix} \right\} = 0$ , and we want the recurrence relations of equation 1.8.1 and in theorem 1.8.3 to be true. Show that under these conditions there is a unique way to extend the definitions to all integers, and that when this is done,  $\left\{ \begin{smallmatrix} n \\ k \end{smallmatrix} \right\} = \left[ \begin{smallmatrix} -k \\ -n \end{smallmatrix} \right]$  for all integers  $n$  and  $k$ . Thus, the extended table of values for either  $\left[ \begin{smallmatrix} n \\ k \end{smallmatrix} \right]$  or  $\left\{ \begin{smallmatrix} n \\ k \end{smallmatrix} \right\}$  will contain all the values of both  $\left[ \begin{smallmatrix} n \\ k \end{smallmatrix} \right]$  and  $\left\{ \begin{smallmatrix} n \\ k \end{smallmatrix} \right\}$ .
11. Under the assumptions that  $s(n, 0) = s(0, n) = 0$  for  $n \neq 0$ , and  $s(n, k) = s(n-1, k-1) - (n-1)s(n-1, k)$ , extend the table for  $s(n, k)$  to all integers, and find a connection to  $S(n, k)$  similar to that in the previous problem.
12. Prove corollary 1.8.4.
13. Prove the remaining part of theorem 1.8.6.



# 2

## Inclusion-Exclusion

### 2.1 THE INCLUSION-EXCLUSION FORMULA

Let's return to a problem we have mentioned but not solved:

**EXAMPLE 2.1.1** How many submultisets of the multiset  $\{2 \cdot a, 4 \cdot b, 3 \cdot c\}$  have size 7?

We recast the problem: this is the number of solutions to  $x_1 + x_2 + x_3 = 7$  with  $0 \leq x_1 \leq 2$ ,  $0 \leq x_2 \leq 4$ ,  $0 \leq x_3 \leq 3$ . We know that the number of solutions in non-negative integers is  $\binom{7+3-1}{3-1} = \binom{9}{2}$ , so this is an overcount, since we count solutions that do not meet the upper bound restrictions. For example, this includes some solutions with  $x_1 \geq 3$ ; how many of these are there? This is a problem we can solve: it is the number of solutions to  $x_1 + x_2 + x_3 = 7$  with  $3 \leq x_1$ ,  $0 \leq x_2$ ,  $0 \leq x_3$ . This is the same as the number of non-negative solutions of  $y_1 + y_2 + y_3 = 7 - 3 = 4$ , or  $\binom{4+3-1}{3-1} = \binom{6}{2}$ . Thus,  $\binom{9}{2} - \binom{6}{2}$  corrects this overcount. If we likewise correct for the overcounting of solutions with  $x_2 \geq 5$  and  $x_3 \geq 4$ , we get  $\binom{9}{2} - \binom{6}{2} - \binom{4}{2} - \binom{5}{2}$ . Is this correct? Not necessarily, because we now have a potential undercount: we have twice subtracted 1 for a solution in which both  $x_1 \geq 3$  and  $x_2 \geq 5$ , when we should have subtracted just 1. However, by good fortune, there are no such solutions, since  $3 + 5 > 7$ . But the same applies to the other pairs of variables: How many solutions have  $x_1 \geq 3$  and  $x_3 \geq 4$ ? It's easy to see there is only one such solution, namely  $3 + 0 + 4 = 7$ . Finally, there are no solutions with  $x_2 \geq 5$  and  $x_3 \geq 4$ , so the corrected count is now  $\binom{9}{2} - \binom{6}{2} - \binom{4}{2} - \binom{5}{2} + 1$ . This does not take into account any solutions in which  $x_1 \geq 3$ ,  $x_2 \geq 5$ , and  $x_3 \geq 4$ , but there are none of these, so

the actual count is

$$\binom{9}{2} - \binom{6}{2} - \binom{4}{2} - \binom{5}{2} + 1 = 36 - 15 - 6 - 10 + 1 = 6.$$

This is small enough that it is not hard to verify by listing all the solutions.  $\square$

So we solved this problem, but it is apparent that it could have been much worse, if the number of variables were larger and there were many complicated overcounts and undercounts. Remarkably, it is possible to streamline this sort of argument; it will still, often, be quite messy, but the reasoning will be simpler.

Let's start by rephrasing the example. Let  $S$  be the set of all non-negative solutions to  $x_1 + x_2 + x_3 = 7$ , let  $A_1$  be all solutions with  $x_1 \geq 3$ ,  $A_2$  all solutions with  $x_2 \geq 5$ , and  $A_3$  all solutions with  $x_3 \geq 4$ . We want to know the size of  $A_1^c \cap A_2^c \cap A_3^c$ , the solutions for which it is not true that  $x_1 \geq 3$  and not true that  $x_2 \geq 5$  and not true that  $x_3 \geq 4$ . Examining our solution, we see that the final count is

$$\begin{aligned} |S| - |A_1| - |A_2| - |A_3| + |A_1 \cap A_2| + |A_1 \cap A_3| + |A_2 \cap A_3| - |A_1 \cap A_2 \cap A_3| \\ = 36 - 15 - 6 - 10 + 0 + 1 + 0 - 0. \end{aligned}$$

This pattern is completely general:

**THEOREM 2.1.2 The inclusion-exclusion formula** If  $A_i \subseteq S$  for  $1 \leq i \leq n$  then

$$|A_1^c \cap \cdots \cap A_n^c| = |S| - |A_1| - \cdots - |A_n| + |A_1 \cap A_2| + \cdots - |A_1 \cap A_2 \cap A_3| - \cdots,$$

or more compactly:

$$|\bigcap_{i=1}^n A_i^c| = |S| + \sum_{k=1}^n (-1)^k \sum |\bigcap_{j=1}^k A_{i_j}|,$$

where the internal sum is over all subsets  $\{i_1, i_2, \dots, i_k\}$  of  $\{1, 2, \dots, n\}$ . Alternately we may write

$$|\bigcap_{i=1}^n A_i^c| = |S| + \sum_{k=1}^n \sum_{I \subseteq [n]} (-1)^{|I|} |\bigcap_{j \in I} A_j|.$$

**Proof.** We need to show that each element of  $\bigcap_{i=1}^n A_i^c$  is counted once by the right hand side, and every other element of  $S$  is counted zero times. The first of these is easy: if  $x \in \bigcap_{i=1}^n A_i^c$  then for every  $i$ ,  $x \notin A_i$ , so  $x$  is in none of the sets involving the  $A_i$  on the right hand side, and so  $x$  is counted, once, by the term  $|S|$ .

Now suppose  $x \notin \bigcap_{i=1}^n A_i^c$ . On the right hand side,  $x$  is counted once by the term  $|S|$ . For some values  $i_1, i_2, \dots, i_k$ ,  $x \in A_{i_m}$ ,  $1 \leq m \leq k$ , and  $x$  is not in the remaining sets

$A_i$ . Then  $x$  is counted zero times by any term involving an  $A_i$  with  $i \notin \{i_1, i_2, \dots, i_k\}$ , and is counted once, positively or negatively, by each term involving only  $A_{i_1}, A_{i_2}, \dots, A_{i_k}$ . There are  $k$  terms of the form  $-|A_{i_m}|$ , which count  $x$  a total of  $-k$  times. There are  $\binom{k}{2}$  terms of the form  $|A_{i_l} \cap A_{i_m}|$ , counting  $x$  a total of  $\binom{k}{2}$  times. Continuing in this way, we see that the final count for  $x$  on the right hand side is

$$1 - k + \binom{k}{2} - \binom{k}{3} + \cdots + (-1)^k \binom{k}{k},$$

or more compactly

$$\sum_{i=0}^k (-1)^i \binom{k}{i}.$$

We know that this alternating sum of binomial coefficients is zero, so  $x$  is counted zero times, as desired. (See equation 1.3.1.) ■

An alternate form of the inclusion exclusion formula is sometimes useful.

**COROLLARY 2.1.3** If  $A_i \subseteq S$  for  $1 \leq i \leq n$  then

$$|\bigcup_{i=1}^n A_i| = \sum_{k=1}^n (-1)^{k+1} \sum |\bigcap_{j=1}^k A_{i_j}|,$$

where the internal sum is over all subsets  $\{i_1, i_2, \dots, i_k\}$  of  $\{1, 2, \dots, n\}$ .

**Proof.** Since  $(\bigcup_{i=1}^n A_i)^c = \bigcap_{i=1}^n A_i^c$ ,

$$\begin{aligned} |\bigcup_{i=1}^n A_i| &= |S| - |\bigcap_{i=1}^n A_i^c| \\ &= |S| - (|S| + \sum_{k=1}^n (-1)^k \sum |\bigcap_{j=1}^k A_{i_j}|) \\ &= (-1) \sum_{k=1}^n (-1)^k \sum |\bigcap_{j=1}^k A_{i_j}| \\ &= \sum_{k=1}^n (-1)^{k+1} \sum |\bigcap_{j=1}^k A_{i_j}|. \end{aligned}$$

Since the right hand side of the inclusion-exclusion formula consists of  $2^n$  terms to be added, it can still be quite tedious. In some nice cases, all intersections of the same

number of sets have the same size. Since there are  $\binom{n}{k}$  possible intersections consisting of  $k$  sets, the formula becomes

$$|\bigcap_{i=1}^n A_i^c| = |S| + \sum_{k=1}^n (-1)^k \binom{n}{k} m_k, \quad (2.1.1)$$

where  $m_k$  is the size of an intersection of  $k$  of the sets.

**EXAMPLE 2.1.4** Find the number of solutions to  $x_1 + x_2 + x_3 + x_4 = 25$ ,  $0 \leq x_i \leq 10$ . Let  $A_i$  be the solutions of  $x_1 + x_2 + x_3 + x_4 = 25$  with  $x_i \geq 11$ . The number of solutions with  $x_i \geq 0$  for all  $i$  is  $\binom{25+4-1}{4-1} = \binom{25+3}{3}$ . Also  $|A_i| = \binom{14+3}{3}$ , and  $|A_i \cap A_j| = \binom{3+3}{3}$ . There are no solutions with 3 or 4 of the variables larger than 10. Hence the number of solutions is

$$\binom{25+3}{3} - \binom{4}{1} \binom{14+3}{3} + \binom{4}{2} \binom{3+3}{3} = 676.$$

□

### Exercises 2.1.

1. List all 6 solutions to the restricted equation in example 2.1.1, and list the corresponding 6 submultisets.
2. Find the number of integer solutions to  $x_1 + x_2 + x_3 + x_4 = 25$ ,  $1 \leq x_1 \leq 6$ ,  $2 \leq x_2 \leq 8$ ,  $0 \leq x_3 \leq 8$ ,  $5 \leq x_4 \leq 9$ .
3. Find the number of submultisets of  $\{25 \cdot a, 25 \cdot b, 25 \cdot c, 25 \cdot d\}$  of size 80.
4. Recall that  $\{ \binom{n}{k} \}$  is a Stirling number of the second kind (definition 1.8.1). Prove that for  $n \geq k \geq 0$ ,

$$\left\{ \binom{n}{k} \right\} = \frac{1}{k!} \sum_{i=0}^k (-1)^{k-i} i^n \binom{k}{i}.$$

Do  $n = 0$  as a special case, then use inclusion-exclusion for the rest. You may assume, by convention, that  $0^0 = 1$ .

## 2.2 FORBIDDEN POSITION PERMUTATIONS

Suppose we shuffle a deck of cards; what is the probability that no card is in its original location? More generally, how many permutations of  $[n] = \{1, 2, 3, \dots, n\}$  have none of the integers in their “correct” locations? That is, 1 is not first, 2 is not second, and so on. Such a permutation is called a **derangement** of  $[n]$ .

Let  $S$  be the set of all permutations of  $[n]$  and  $A_i$  be the permutations of  $[n]$  in which  $i$  is in the correct place. Then we want to know  $|\bigcap_{i=1}^n A_i^c|$ .

For any  $i$ ,  $|A_i| = (n-1)!$ : once  $i$  is fixed in position  $i$ , the remaining  $n-1$  integers can be placed in any locations.

What about  $|A_i \cap A_j|$ ? If both  $i$  and  $j$  are in the correct position, the remaining  $n - 2$  integers can be placed anywhere, so  $|A_i \cap A_j| = (n - 2)!$ .

In the same way, we see that  $|A_{i_1} \cap A_{i_2} \cap \dots \cap A_{i_k}| = (n - k)!$ . Thus, by the inclusion-exclusion formula, in the form of equation 2.1.1,

$$\begin{aligned} |\bigcap_{i=1}^n A_i^c| &= |S| + \sum_{k=1}^n (-1)^k \binom{n}{k} (n-k)! \\ &= n! + \sum_{k=1}^n (-1)^k \frac{n!}{k!(n-k)!} (n-k)! \\ &= n! + \sum_{k=1}^n (-1)^k \frac{n!}{k!} \\ &= n! + n! \sum_{k=1}^n (-1)^k \frac{1}{k!} \\ &= n! \left(1 + \sum_{k=1}^n (-1)^k \frac{1}{k!}\right) \\ &= n! \sum_{k=0}^n (-1)^k \frac{1}{k!}. \end{aligned}$$

The last sum should look familiar:

$$e^x = \sum_{k=0}^{\infty} \frac{1}{k!} x^k.$$

Substituting  $x = -1$  gives

$$e^{-1} = \sum_{k=0}^{\infty} \frac{1}{k!} (-1)^k.$$

The probability of getting a derangement by chance is then

$$\frac{1}{n!} n! \sum_{k=0}^n (-1)^k \frac{1}{k!} = \sum_{k=0}^n (-1)^k \frac{1}{k!},$$

and when  $n$  is bigger than 6, this is quite close to

$$e^{-1} \approx 0.3679.$$

So in the case of a deck of cards, the probability of a derangement is about 37%.

Let  $D_n = n! \sum_{k=0}^n (-1)^k \frac{1}{k!}$ . These **derangement numbers** have some interesting properties. First, note that when  $n = 0$ , we have  $D_0 = 0!(-1)^0 \frac{1}{0!} = 1$ . “Derangements

## 50 Chapter 2 Inclusion-Exclusion

of the empty set” doesn’t really make sense, but it is useful to adopt the convention that  $D_0 = 1$ .

The derangements of  $[n]$  may be produced as follows: For each  $i \in \{2, 3, \dots, n\}$ , put  $i$  in position 1 and 1 in position  $i$ . Then permute the numbers  $\{2, 3, \dots, i-1, i+1, \dots, n\}$  in all possible ways so that none of these  $n-2$  numbers is in the correct place. There are  $D_{n-2}$  ways to do this. Then, keeping 1 in position  $i$ , derange the numbers  $\{i, 2, 3, \dots, i-1, i+1, \dots, n\}$ , with the “correct” position of  $i$  now considered to be position 1. There are  $D_{n-1}$  ways to do this. Thus,  $D_n = (n-1)(D_{n-1} + D_{n-2})$ . Starting with  $D_0 = 1$  and  $D_1 = 0$ , this gives  $D_2 = (1)(0+1) = 1$  and  $D_3 = (2)(1+0) = 2$ , both of which are easy to check directly.

We explore this **recurrence relation** a bit:

$$\begin{aligned}
 D_n &= nD_{n-1} - D_{n-1} + (n-1)D_{n-2} & (*) \\
 &= nD_{n-1} - (n-2)(D_{n-2} + D_{n-3}) + (n-1)D_{n-2} \\
 &= nD_{n-1} - (n-2)D_{n-2} - (n-2)D_{n-3} + (n-1)D_{n-2} \\
 &= nD_{n-1} + D_{n-2} - (n-2)D_{n-3} & (*) \\
 &= nD_{n-1} + (n-3)(D_{n-3} + D_{n-4}) - (n-2)D_{n-3} \\
 &= nD_{n-1} + (n-3)D_{n-3} + (n-3)D_{n-4} - (n-2)D_{n-3} \\
 &= nD_{n-1} - D_{n-3} + (n-3)D_{n-4} & (*) \\
 &= nD_{n-1} - (n-4)(D_{n-4} + D_{n-5}) + (n-3)D_{n-4} \\
 &= nD_{n-1} - (n-4)D_{n-4} - (n-4)D_{n-5} + (n-3)D_{n-4} \\
 &= nD_{n-1} + D_{n-4} - (n-4)D_{n-5}. & (*) \\
 \end{aligned}$$

It appears from the starred lines that the pattern here is that

$$D_n = nD_{n-1} + (-1)^k D_{n-k} + (-1)^{k+1} (n-k) D_{n-k-1}.$$

If this continues, we should get to

$$D_n = nD_{n-1} + (-1)^{n-2} D_2 + (-1)^{n-1} (2) D_1.$$

Since  $D_2 = 1$  and  $D_1 = 0$ , this would give

$$D_n = nD_{n-1} + (-1)^n,$$

since  $(-1)^n = (-1)^{n-2}$ . Indeed this is true, and can be proved by induction. This gives a somewhat simpler recurrence relation, making it quite easy to compute  $D_n$ .

• • •

There are many similar problems.

**EXAMPLE 2.2.1** How many permutations of  $[n]$  contain no instance of  $i$  followed by  $i + 1$ ?

By a similar use of the inclusion-exclusion formula, it turns out that this is

$$Q_n = n! \sum_{k=0}^{n-1} (-1)^k \frac{1}{k!} + (n-1)! \sum_{k=1}^{n-1} (-1)^{k-1} \frac{1}{(k-1)!}.$$

Note that the limits on the two sums are not identical.  $\square$

### Exercises 2.2.

1. Prove that  $D_n = nD_{n-1} + (-1)^n$  when  $n \geq 2$ , by induction on  $n$ .
2. Prove that  $D_n$  is even if and only if  $n$  is odd.
3. Provide the missing details for example 2.2.1. What is  $\lim_{n \rightarrow \infty} \frac{Q_n}{n!}$ ?
4. Find the number of permutations of  $1, 2, \dots, 8$  that have no odd number in the correct position.
5. Find the number of permutations of  $1, 2, \dots, 8$  that have at least one odd number in the correct position.
6. How many permutations of  $[n]$  have exactly  $k$  numbers in their correct positions?
7. Give a combinatorial proof that

$$n! = \sum_{k=0}^n \binom{n}{k} D_{n-k}.$$

8. A small merry-go-round has 8 seats occupied by 8 children. In how many ways can the children change places so that no child sits behind the same child as on the first ride? The seats do not matter, only the relative positions of the children.
9. On the way into a party everyone checks a coat and a bag at the door. On the way out, the attendant hands out coats and bags randomly. In how many ways can this be done if
  - (a) No one gets either their own coat or their own bag?
  - (b) One may get one's own coat, or bag, but not both.
10. Suppose  $n$  people are seated in  $m \geq n$  chairs in a room. At some point there is a break, and everyone leaves the room. When they return, in how many ways can they be seated so that no person occupies the same chair as before the break?



# 3

## Generating Functions

As we have seen, a typical counting problem includes one or more parameters, which of course show up in the solutions, such as  $\binom{n}{k}$ ,  $P(n, k)$ , or the number of derangements of  $[n]$ . Also recall that

$$(x + 1)^n = \sum_{k=0}^n \binom{n}{k} x^k.$$

This provides the values  $\binom{n}{k}$  as coefficients of the Maclaurin expansion of a function. This turns out to be a useful idea.

**DEFINITION 3.0.1**  $f(x)$  is a **generating function** for the sequence  $a_0, a_1, a_2, \dots$  if

$$f(x) = \sum_{i=0}^{\infty} a_i x^i.$$

□

Sometimes a generating function can be used to find a formula for its coefficients, but if not, it gives a way to generate them. Generating functions can also be useful in proving facts about the coefficients.

### 3.1 NEWTON'S BINOMIAL THEOREM

Recall that

$$\binom{n}{k} = \frac{n!}{k! (n - k)!} = \frac{n(n - 1)(n - 2) \cdots (n - k + 1)}{k!}.$$

## 54 Chapter 3 Generating Functions

The expression on the right makes sense even if  $n$  is not a non-negative integer, so long as  $k$  is a non-negative integer, and we therefore define

$$\binom{r}{k} = \frac{r(r-1)(r-2)\cdots(r-k+1)}{k!}$$

when  $r$  is a real number. For example,

$$\binom{1/2}{4} = \frac{(1/2)(-1/2)(-3/2)(-5/2)}{4!} = \frac{-5}{128} \quad \text{and} \quad \binom{-2}{3} = \frac{(-2)(-3)(-4)}{3!} = -4.$$

These **generalized binomial coefficients** share some important properties of the usual binomial coefficients, most notably that

$$\binom{r}{k} = \binom{r-1}{k-1} + \binom{r-1}{k}. \quad (3.1.1)$$

Then remarkably:

**THEOREM 3.1.1 Newton's Binomial Theorem** For any real number  $r$  that is not a non-negative integer,

$$(x+1)^r = \sum_{i=0}^{\infty} \binom{r}{i} x^i$$

when  $-1 < x < 1$ .

**Proof.** It is not hard to see that the series is the Maclaurin series for  $(x+1)^r$ , and that the series converges when  $-1 < x < 1$ . It is rather more difficult to prove that the series is equal to  $(x+1)^r$ ; the proof may be found in many introductory real analysis books. ■

**EXAMPLE 3.1.2** Expand the function  $(1-x)^{-n}$  when  $n$  is a positive integer.

We first consider  $(x+1)^{-n}$ ; we can simplify the binomial coefficients:

$$\begin{aligned} \frac{(-n)(-n-1)(-n-2)\cdots(-n-i+1)}{i!} &= (-1)^i \frac{(n)(n+1)\cdots(n+i-1)}{i!} \\ &= (-1)^i \frac{(n+i-1)!}{i!(n-1)!} \\ &= (-1)^i \binom{n+i-1}{i} = (-1)^i \binom{n+i-1}{n-1}. \end{aligned}$$

Thus

$$(x+1)^{-n} = \sum_{i=0}^{\infty} (-1)^i \binom{n+i-1}{n-1} x^i = \sum_{i=0}^{\infty} \binom{n+i-1}{n-1} (-x)^i.$$

Now replacing  $x$  by  $-x$  gives

$$(1-x)^{-n} = \sum_{i=0}^{\infty} \binom{n+i-1}{n-1} x^i.$$

So  $(1-x)^{-n}$  is the generating function for  $\binom{n+i-1}{n-1}$ , the number of submultisets of  $\{\infty \cdot 1, \infty \cdot 2, \dots, \infty \cdot n\}$  of size  $i$ . □

In many cases it is possible to directly construct the generating function whose coefficients solve a counting problem.

**EXAMPLE 3.1.3** Find the number of solutions to  $x_1 + x_2 + x_3 + x_4 = 17$ , where  $0 \leq x_1 \leq 2$ ,  $0 \leq x_2 \leq 5$ ,  $0 \leq x_3 \leq 5$ ,  $2 \leq x_4 \leq 6$ .

We can of course solve this problem using the inclusion-exclusion formula, but we use generating functions. Consider the function

$$(1 + x + x^2)(1 + x + x^2 + x^3 + x^4 + x^5)(1 + x + x^2 + x^3 + x^4 + x^5)(x^2 + x^3 + x^4 + x^5 + x^6).$$

We can multiply this out by choosing one term from each factor in all possible ways. If we then collect like terms, the coefficient of  $x^k$  will be the number of ways to choose one term from each factor so that the exponents of the terms add up to  $k$ . This is precisely the number of solutions to  $x_1 + x_2 + x_3 + x_4 = k$ , where  $0 \leq x_1 \leq 2$ ,  $0 \leq x_2 \leq 5$ ,  $0 \leq x_3 \leq 5$ ,  $2 \leq x_4 \leq 6$ . Thus, the answer to the problem is the coefficient of  $x^{17}$ . With the help of a computer algebra system we get

$$\begin{aligned} & (1 + x + x^2)(1 + x + x^2 + x^3 + x^4 + x^5)^2(x^2 + x^3 + x^4 + x^5 + x^6) \\ &= x^{18} + 4x^{17} + 10x^{16} + 19x^{15} + 31x^{14} + 45x^{13} + 58x^{12} + 67x^{11} + 70x^{10} \\ &\quad + 67x^9 + 58x^8 + 45x^7 + 31x^6 + 19x^5 + 10x^4 + 4x^3 + x^2, \end{aligned}$$

so the answer is 4. □

**EXAMPLE 3.1.4** Find the generating function for the number of solutions to  $x_1 + x_2 + x_3 + x_4 = k$ , where  $0 \leq x_1 \leq \infty$ ,  $0 \leq x_2 \leq 5$ ,  $0 \leq x_3 \leq 5$ ,  $2 \leq x_4 \leq 6$ .

This is just like the previous example except that  $x_1$  is not bounded above. The generating function is thus

$$\begin{aligned} f(x) &= (1 + x + x^2 + \dots)(1 + x + x^2 + x^3 + x^4 + x^5)^2(x^2 + x^3 + x^4 + x^5 + x^6) \\ &= (1 - x)^{-1}(1 + x + x^2 + x^3 + x^4 + x^5)^2(x^2 + x^3 + x^4 + x^5 + x^6) \\ &= \frac{(1 + x + x^2 + x^3 + x^4 + x^5)^2(x^2 + x^3 + x^4 + x^5 + x^6)}{1 - x}. \end{aligned}$$

Note that  $(1 - x)^{-1} = (1 + x + x^2 + \dots)$  is the familiar geometric series from calculus; alternately, we could use example 3.1.2. Unlike the function in the previous example, this function has an infinite expansion:

$$\begin{aligned} f(x) &= x^2 + 4x^3 + 10x^4 + 20x^5 + 35x^6 + 55x^7 + 78x^8 \\ &\quad + 102x^9 + 125x^{10} + 145x^{11} + 160x^{12} + 170x^{13} + 176x^{14} \\ &\quad + 179x^{15} + 180x^{16} + 180x^{17} + 180x^{18} + 180x^{19} + 180x^{20} + \dots. \end{aligned}$$

You can see how to do this in Sage. □

**EXAMPLE 3.1.5** Find a generating function for the number of submultisets of  $\{\infty \cdot a, \infty \cdot b, \infty \cdot c\}$  in which there are an odd number of  $as$ , an even number of  $bs$ , and any number of  $cs$ . As we have seen, this is the same as the number of solutions to  $x_1 + x_2 + x_3 = n$  in which  $x_1$  is odd,  $x_2$  is even, and  $x_3$  is unrestricted. The generating function is therefore

$$\begin{aligned} & (x + x^3 + x^5 + \dots)(1 + x^2 + x^4 + \dots)(1 + x + x^2 + x^3 + \dots) \\ &= x(1 + (x^2) + (x^2)^2 + (x^2)^3 + \dots)(1 + (x^2) + (x^2)^2 + (x^2)^3 + \dots) \frac{1}{1-x} \\ &= \frac{x}{(1-x^2)^2(1-x)}. \end{aligned}$$

□

### Exercises 3.1.

For some of these exercises, you may want to use the sage applet above, in example 3.1.4, or your favorite computer algebra system.

1. Prove that  $\binom{r}{k} = \binom{r-1}{k-1} + \binom{r-1}{k}$ .
2. Show that the Maclaurin series for  $(x+1)^r$  is  $\sum_{i=0}^{\infty} \binom{r}{i} x^i$ .
3. Concerning example 3.1.4, show that all coefficients beginning with  $x^{16}$  are 180.
4. Use a generating function to find the number of solutions to  $x_1 + x_2 + x_3 + x_4 = 14$ , where  $0 \leq x_1 \leq 3, 2 \leq x_2 \leq 5, 0 \leq x_3 \leq 5, 4 \leq x_4 \leq 6$ .
5. Find the generating function for the number of solutions to  $x_1 + x_2 + x_3 + x_4 = k$ , where  $0 \leq x_1 \leq \infty, 3 \leq x_2 \leq \infty, 2 \leq x_3 \leq 5, 1 \leq x_4 \leq 5$ .
6. Find a generating function for the number of non-negative integer solutions to  $3x + 2y + 7z = n$ .
7. Suppose we have a large supply of red, white, and blue balloons. How many different bunches of 10 balloons are there, if each bunch must have at least one balloon of each color and the number of white balloons must be even?
8. Use generating functions to show that every positive integer can be written in exactly one way as a sum of distinct powers of 2.
9. Suppose we have a large supply of blue and green candles, and one gold candle. How many collections of  $n$  candles are there in which the number of blue candles is even, the number of green candles is any number, and the number of gold candles is at most one?

## 3.2 EXPONENTIAL GENERATING FUNCTIONS

There are other ways that a function might be said to generate a sequence, other than as what we have called a generating function. For example,

$$e^x = \sum_{n=0}^{\infty} \frac{1}{n!} x^n$$

is the generating function for the sequence  $1, 1, \frac{1}{2}, \frac{1}{3!}, \dots$ . But if we write the sum as

$$e^x = \sum_{n=0}^{\infty} 1 \cdot \frac{x^n}{n!},$$

considering the  $n!$  to be part of the expression  $x^n/n!$ , we might think of this same function as generating the sequence  $1, 1, 1, \dots$ , interpreting 1 as the coefficient of  $x^n/n!$ . This is not a very interesting sequence, of course, but this idea can often prove fruitful. If

$$f(x) = \sum_{n=0}^{\infty} a_n \frac{x^n}{n!},$$

we say that  $f(x)$  is the **exponential generating function** for  $a_0, a_1, a_2, \dots$ .

**EXAMPLE 3.2.1** Find an exponential generating function for the number of permutations with repetition of length  $n$  of the set  $\{a, b, c\}$ , in which there are an odd number of  $a$ s, an even number of  $b$ s, and any number of  $c$ s.

For a fixed  $n$  and fixed numbers of the letters, we already know how to do this. For example, if we have 3  $a$ s, 4  $b$ s, and 2  $c$ s, there are  $\binom{9}{3 \ 4 \ 5}$  such permutations. Now consider the following function:

$$\sum_{i=0}^{\infty} \frac{x^{2i+1}}{(2i+1)!} \sum_{i=0}^{\infty} \frac{x^{2i}}{(2i)!} \sum_{i=0}^{\infty} \frac{x^i}{i!}.$$

What is the coefficient of  $x^9/9!$  in this product? One way to get an  $x^9$  term is

$$\frac{x^3}{3!} \frac{x^4}{4!} \frac{x^2}{2!} = \frac{9!}{3! 4! 2!} \frac{x^9}{9!} = \binom{9}{3 \ 4 \ 5} \frac{x^9}{9!}.$$

That is, this one term counts the number of permutations in which there are 3  $a$ s, 4  $b$ s, and 2  $c$ s. The ultimate coefficient of  $x^9/9!$  will be the sum of many such terms, counting the contributions of all possible choices of an odd number of  $a$ s, an even number of  $b$ s, and any number of  $c$ s.

Now we notice that  $\sum_{i=0}^{\infty} \frac{x^i}{i!} = e^x$ , and that the other two sums are closely related to this. A little thought leads to

$$e^x + e^{-x} = \sum_{i=0}^{\infty} \frac{x^i}{i!} + \sum_{i=0}^{\infty} \frac{(-x)^i}{i!} = \sum_{i=0}^{\infty} \frac{x^i + (-x)^i}{i!}.$$

Now  $x^i + (-x)^i$  is  $2x^i$  when  $i$  is even, and 0 when  $x$  is odd. Thus

$$e^x + e^{-x} = \sum_{i=0}^{\infty} \frac{2x^{2i}}{(2i)!},$$

## 58 Chapter 3 Generating Functions

so that

$$\sum_{i=0}^{\infty} \frac{x^{2i}}{(2i)!} = \frac{e^x + e^{-x}}{2}.$$

A similar manipulation shows that

$$\sum_{i=0}^{\infty} \frac{x^{2i+1}}{(2i+1)!} = \frac{e^x - e^{-x}}{2}.$$

Thus, the generating function we seek is

$$\frac{e^x - e^{-x}}{2} \cdot \frac{e^x + e^{-x}}{2} e^x = \frac{1}{4} (e^x - e^{-x})(e^x + e^{-x}) e^x = \frac{1}{4} (e^{3x} - e^{-x}).$$

Notice the similarity to example 3.1.5. □

### Exercises 3.2.

1. Find the coefficient of  $x^9/9!$  in the function of example 3.2.1. You may use [Sage](#) or a similar program.
2. Find an exponential generating function for the number of permutations with repetition of length  $n$  of the set  $\{a, b, c\}$ , in which there are an odd number of  $a$ s, an even number of  $b$ s, and an even number of  $c$ s.
3. Find an exponential generating function for the number of permutations with repetition of length  $n$  of the set  $\{a, b, c\}$ , in which the number of  $a$ s is even and at least 2, the number of  $b$ s is even and at most 6, and the number of  $c$ s is at least 3.
4. In how many ways can we paint the 10 rooms of a hotel if at most three can be painted red, at most 2 painted green, at most 1 painted white, and any number can be painted blue or orange?
5. Recall from section 1.4 that the Bell numbers  $B_n$  count all of the partitions of  $\{1, 2, \dots, n\}$ .

Let  $f(x) = \sum_{n=0}^{\infty} B_n \cdot \frac{x^n}{n!}$ , and note that

$$f'(x) = \sum_{n=1}^{\infty} B_n \frac{x^{n-1}}{(n-1)!} = \sum_{n=0}^{\infty} B_{n+1} \frac{x^n}{n!} = \sum_{n=0}^{\infty} \left( \sum_{k=0}^n \binom{n}{k} B_{n-k} \right) \frac{x^n}{n!},$$

using the recurrence relation 1.4.1 for  $B_{n+1}$  from section 1.4. Now it is possible to write this as a product of two infinite series:

$$f'(x) = \left( \sum_{n=0}^{\infty} B_n \cdot \frac{x^n}{n!} \right) \left( \sum_{n=0}^{\infty} a_n x^n \right) = f(x)g(x).$$

Find an expression for  $a_n$  that makes this true, which will tell you what  $g(x)$  is, then solve the differential equation for  $f(x)$ , the exponential generating function for the Bell numbers.

From section 1.4, the first few Bell numbers are 1, 1, 2, 5, 15, 52, 203, 877, 4140, 21147, 115975, 678570, 4213597, 27644437. You can use [Sage](#) to check your answer.

### 3.3 PARTITIONS OF INTEGERS

**DEFINITION 3.3.1** A **partition** of a positive integer  $n$  is a multiset of positive integers that sum to  $n$ . We denote the number of partitions of  $n$  by  $p_n$ .  $\square$

Typically a partition is written as a sum, not explicitly as a multiset. Using the usual convention that an empty sum is 0, we say that  $p_0 = 1$ .

**EXAMPLE 3.3.2** The partitions of 5 are

$$\begin{aligned} & 5 \\ & 4 + 1 \\ & 3 + 2 \\ & 3 + 1 + 1 \\ & 2 + 2 + 1 \\ & 2 + 1 + 1 + 1 \\ & 1 + 1 + 1 + 1 + 1. \end{aligned}$$

Thus  $p_5 = 7$ .  $\square$

There is no simple formula for  $p_n$ , but it is not hard to find a generating function for them. As with some previous examples, we seek a product of factors so that when the factors are multiplied out, the coefficient of  $x^n$  is  $p_n$ . We would like each  $x^n$  term to represent a single partition, before like terms are collected. A partition is uniquely described by the number of 1s, number of 2s, and so on, that is, by the repetition numbers of the multiset. We devote one factor to each integer:

$$(1 + x + x^2 + x^3 + \dots)(1 + x^2 + x^4 + x^6 + \dots) \cdots (1 + x^k + x^{2k} + x^{3k} + \dots) \cdots = \prod_{k=1}^{\infty} \sum_{i=0}^{\infty} x^{ik}.$$

When this product is expanded, we pick one term from each factor in all possible ways, with the further condition that we only pick a finite number of “non-1” terms. For example, if we pick  $x^3$  from the first factor,  $x^3$  from the third factor,  $x^{15}$  from the fifth factor, and 1s from all other factors, we get  $x^{21}$ . In the context of the product, this represents  $3 \cdot 1 + 1 \cdot 3 + 3 \cdot 5$ , corresponding to the partition  $1 + 1 + 1 + 3 + 5 + 5 + 5$ , that is, three 1s, one 3, and three 5s. Each factor is a geometric series; the  $k$ th factor is

$$1 + x^k + (x^k)^2 + (x^k)^3 + \cdots = \frac{1}{1 - x^k},$$

so the generating function can be written

$$\prod_{k=1}^{\infty} \frac{1}{1 - x^k}.$$

## 60 Chapter 3 Generating Functions

Note that if we are interested in some particular  $p_n$ , we do not need the entire infinite product, or even any complete factor, since no partition of  $n$  can use any integer greater than  $n$ , and also cannot use more than  $n/k$  copies of  $k$ .

**EXAMPLE 3.3.3** Find  $p_8$ .

We expand

$$\begin{aligned} & (1 + x + x^2 + x^3 + x^4 + x^5 + x^6 + x^7 + x^8)(1 + x^2 + x^4 + x^6 + x^8)(1 + x^3 + x^6) \\ & \quad (1 + x^4 + x^8)(1 + x^5)(1 + x^6)(1 + x^7)(1 + x^8) \\ & = 1 + x + 2x^2 + 3x^3 + 5x^4 + 7x^5 + 11x^6 + 15x^7 + 22x^8 + \cdots + x^{56}, \end{aligned}$$

so  $p_8 = 22$ . Note that all of the coefficients prior to this are also correct, but the following coefficients are not necessarily the corresponding partition numbers.  $\square$

Partitions of integers have some interesting properties. Let  $p_d(n)$  be the number of partitions of  $n$  into distinct parts; let  $p_o(n)$  be the number of partitions into odd parts.

**EXAMPLE 3.3.4** For  $n = 6$ , the partitions into distinct parts are

$$6, 5 + 1, 4 + 2, 3 + 2 + 1,$$

so  $p_d(6) = 4$ , and the partitions into odd parts are

$$5 + 1, 3 + 3, 3 + 1 + 1 + 1, 1 + 1 + 1 + 1 + 1,$$

so  $p_o(6) = 4$ .  $\square$

In fact, for every  $n$ ,  $p_d(n) = p_o(n)$ , and we can see this by manipulating generating functions. The generating function for  $p_d(n)$  is

$$f_d(x) = (1 + x)(1 + x^2)(1 + x^3) \cdots = \prod_{i=1}^{\infty} (1 + x^i).$$

The generating function for  $p_o(n)$  is

$$f_o(x) = (1 + x + x^2 + x^3 + \cdots)(1 + x^3 + x^6 + x^9 + \cdots) \cdots = \prod_{i=0}^{\infty} \frac{1}{1 - x^{2i+1}}.$$

We can write

$$f_d(x) = \frac{1 - x^2}{1 - x} \cdot \frac{1 - x^4}{1 - x^2} \cdot \frac{1 - x^6}{1 - x^3} \cdots$$

and notice that every numerator is eventually canceled by a denominator, leaving only the denominators containing odd powers of  $x$ , so  $f_d(x) = f_o(x)$ .

We can also use a recurrence relation to find the partition numbers, though in a somewhat less direct way than the binomial coefficients or the Bell numbers. Let  $p_k(n)$  be the number of partitions of  $n$  into exactly  $k$  parts. We will find a recurrence relation to compute the  $p_k(n)$ , and then

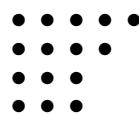
$$p_n = \sum_{k=1}^n p_k(n).$$

Now consider the partitions of  $n$  into  $k$  parts. Some of these partitions contain no 1s, like  $3+3+4+6$ , a partition of 16 into 4 parts. Subtracting 1 from each part, we get a partition of  $n-k$  into  $k$  parts; for the example, this is  $2+2+3+5$ . The remaining partitions of  $n$  into  $k$  parts contain a 1. If we remove the 1, we are left with a partition of  $n-1$  into  $k-1$  parts. This gives us a 1-1 correspondence between the partitions of  $n$  into  $k$  parts, and the partitions of  $n-k$  into  $k$  parts together with the partitions of  $n-1$  into  $k-1$  parts, so  $p_k(n) = p_k(n-k) + p_{k-1}(n-1)$ .

Using this recurrence we can build a triangle containing the  $p_k(n)$ , and the row sums of this triangle give the partition numbers. For all  $n$ ,  $p_1(n) = 1$ , which gives the first column of the triangle, after which the recurrence applies. Also, note that  $p_k(n) = 0$  when  $k > n$  and we let  $p_k(0) = 0$ ; these are needed in some cases to compute the  $p_k(n-k)$  term of the recurrence. Here are the first few rows of the triangle; at the left are the row numbers, and at the right are the row sums, that is, the partition numbers. For the last row, each entry is the sum of the like-colored numbers in the previous rows. Note that beginning with  $p_4(7) = 3$  in the last row,  $p_k(7) = p_{k-1}(6)$ , as  $p_k(7-k) = 0$ .

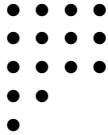
1	1		0	1
2	1	1	0	2
3	1	1	1	3
4	1	2	1	5
5	1	2	2	7
6	1	3	3	11
7	1	3	4	15

Yet another sometimes useful way to think of a partition is with a **Ferrers diagram**. Each integer in the partition is represented by a row of dots, and the rows are ordered from longest on the top to shortest at the bottom. For example, the partition  $3+3+4+5$  would be represented by



The **conjugate** of a partition is the one corresponding to the Ferrers diagram produced by flipping the diagram for the original partition across the main diagonal, thus turning

rows into columns and vice versa. For the diagram above, the conjugate is



with corresponding partition  $1 + 2 + 4 + 4 + 4$ . This concept can occasionally make facts about partitions easier to see than otherwise. Here is a classic example: the number of partitions of  $n$  with largest part  $k$  is the same as the number of partitions into  $k$  parts,  $p_k(n)$ . The action of conjugation takes every partition of one type into a partition of the other: the conjugate of a partition into  $k$  parts is a partition with largest part  $k$  and vice versa. This establishes a 1–1 correspondence between partitions into  $k$  parts and partitions with largest part  $k$ .

### **Exercises 3.3.**

1. Use generating functions to find  $p_{15}$ .
2. Find the generating function for the number of partitions of an integer into distinct odd parts. Find the number of such partitions of 20.
3. Find the generating function for the number of partitions of an integer into distinct even parts. Find the number of such partitions of 30.
4. Find the number of partitions of 25 into odd parts.
5. Find the generating function for the number of partitions of an integer into  $k$  parts; that is, the coefficient of  $x^n$  is the number of partitions of  $n$  into  $k$  parts.
6. Complete row 8 of the table for the  $p_k(n)$ , and verify that the row sum is 22, as we saw in example 3.3.3.
7. A partition of  $n$  is self-conjugate if its Ferrers diagram is symmetric around the main diagonal, so that its conjugate is itself. Show that the number of self-conjugate partitions of  $n$  is equal to the number of partitions of  $n$  into distinct odd parts.

## **3.4 RECURRENCE RELATIONS**

A **recurrence relation** defines a sequence  $\{a_i\}_{i=0}^{\infty}$  by expressing a typical term  $a_n$  in terms of earlier terms,  $a_i$  for  $i < n$ . For example, the famous Fibonacci sequence is defined by

$$F_0 = 0, F_1 = 1, F_n = F_{n-1} + F_{n-2}.$$

Note that some initial values must be specified for the recurrence relation to define a unique sequence.

The starting index for the sequence need not be zero if it doesn't make sense or some other starting index is more convenient. We saw two recurrence relations for the number

of derangements of  $[n]$ :

$$D_1 = 0, D_n = nD_{n-1} + (-1)^n.$$

and

$$D_1 = 0, D_2 = 1, D_n = (n-1)(D_{n-1} + D_{n-2}).$$

To “solve” a recurrence relation means to find a formula for  $a_n$ . There are a variety of methods for solving recurrence relations, with various advantages and disadvantages in particular cases. One method that works for some recurrence relations involves generating functions. The idea is simple, if the execution is not always: Let

$$f(x) = \sum_{i=0}^{\infty} a_i x^i,$$

that is, let  $f(x)$  be the generating function for  $\{a_i\}_{i=0}^{\infty}$ . We now try to manipulate  $f(x)$ , using the recurrence relation, until we can solve for  $f(x)$  explicitly. Finally, we hope that we can find a formula for the coefficients from the formula for  $f(x)$ .

**EXAMPLE 3.4.1** Solve  $F_0 = 0, F_1 = 1, F_n = F_{n-1} + F_{n-2}$ .

Let

$$f(x) = \sum_{i=0}^{\infty} F_i x^i$$

and note that

$$xf(x) = \sum_{i=0}^{\infty} F_i x^{i+1} = \sum_{i=1}^{\infty} F_{i-1} x^i.$$

To get the second sum we have simply “re-indexed” so that the index value gives the exponent on  $x$ , just as in the series for  $f(x)$ . Likewise,

$$x^2 f(x) = \sum_{i=0}^{\infty} F_i x^{i+2} = \sum_{i=2}^{\infty} F_{i-2} x^i.$$

In somewhat more suggestive form, we have

$$\begin{aligned} f(x) &= x + F_2 x^2 + F_3 x^3 + F_4 x^4 + \dots \\ xf(x) &= \qquad x^2 + F_2 x^3 + F_3 x^4 + \dots \\ x^2 f(x) &= \qquad \qquad \qquad x^3 + F_2 x^4 + \dots \end{aligned}$$

and combining the three equations we get

$$f(x) - xf(x) - x^2 f(x) = x + (F_2 - 1)x^2 + (F_3 - F_2 - 1)x^3 + (F_4 - F_3 - F_2)x^4 + \dots$$

## 64 Chapter 3 Generating Functions

or in more compact form

$$\begin{aligned}
f(x) - xf(x) - x^2 f(x) &= \sum_{i=0}^{\infty} F_i x^i - \sum_{i=1}^{\infty} F_{i-1} x^i - \sum_{i=2}^{\infty} F_{i-2} x^i \\
&= x + \sum_{i=2}^{\infty} (F_i - F_{i-1} - F_{i-2}) x^i \\
&= x + \sum_{i=2}^{\infty} 0 \cdot x^i \\
&= x,
\end{aligned}$$

recalling that  $F_0 = 0$  and  $F_1 = 1$ . Now

$$f(x) = \frac{x}{1-x-x^2} = \frac{-x}{x^2+x-1}.$$

If we can find an explicit representation for the series for this function, we will have solved the recurrence relation. Here is where things could go wrong, but in this case it works out. Let  $a$  and  $b$  be the roots of  $x^2 + x - 1$ ; using the quadratic formula, we get

$$a = \frac{-1 + \sqrt{5}}{2}, b = \frac{-1 - \sqrt{5}}{2}.$$

Borrowing a technique from calculus, we write

$$\frac{-x}{x^2+x-1} = \frac{A}{x-a} + \frac{B}{x-b}.$$

Solving for  $A$  and  $B$  gives

$$A = \frac{1 - \sqrt{5}}{2\sqrt{5}}, B = \frac{-1 - \sqrt{5}}{2\sqrt{5}}.$$

Then

$$\frac{-x}{x^2+x-1} = -\frac{A}{a} \frac{1}{1-x/a} - \frac{B}{b} \frac{1}{1-x/b}.$$

From calculus we know that

$$\frac{1}{1-x/a} = \sum_{i=0}^{\infty} (1/a)^i x^i \quad \text{and} \quad \frac{1}{1-x/b} = \sum_{i=0}^{\infty} (1/b)^i x^i.$$

Finally, this means the coefficient of  $x^i$  in the series for  $f(x)$  is

$$F_i = -\frac{A}{a} (1/a)^i - \frac{B}{b} (1/b)^i.$$

Simplifying gives

$$F_i = \frac{1}{\sqrt{5}} \left( \frac{1+\sqrt{5}}{2} \right)^i - \frac{1}{\sqrt{5}} \left( \frac{1-\sqrt{5}}{2} \right)^i.$$

Here's an interesting feature of this expression: since  $|(1-\sqrt{5})/2| < 1$ , the limit of  $((1-\sqrt{5})/2)^i$  as  $i$  goes to infinity is 0. So when  $i$  is large enough,

$$F_i = \text{round} \left( \frac{1}{\sqrt{5}} \left( \frac{1+\sqrt{5}}{2} \right)^i \right),$$

that is, the first term rounded to the nearest integer. As it turns out, this is true starting with  $i = 0$ .

You can see how to do the entire solution in [Sage](#). □

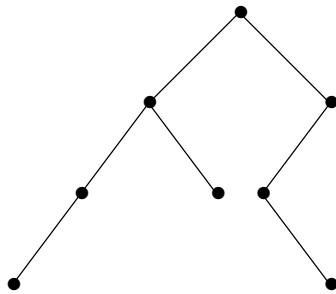
### **Exercises 3.4.**

1. Find the generating function for the solutions to  $h_n = 4h_{n-1} - 3h_{n-2}$ ,  $h_0 = 2$ ,  $h_1 = 5$ , and use it to find a formula for  $h_n$ .
2. Find the generating function for the solutions to  $h_n = 3h_{n-1} + 4h_{n-2}$ ,  $h_0 = h_1 = 1$ , and use it to find a formula for  $h_n$ .
3. Find the generating function for the solutions to  $h_n = 2h_{n-1} + 3^n$ ,  $h_0 = 0$ , and use it to find a formula for  $h_n$ .
4. Find the generating function for the solutions to  $h_n = 4h_{n-2}$ ,  $h_0 = 0$ ,  $h_1 = 1$ , and use it to find a formula for  $h_n$ . (It is easy to discover this formula directly; the point here is to see that the generating function approach gives the correct answer.)
5. Find the generating function for the solutions to  $h_n = h_{n-1} + h_{n-2}$ ,  $h_0 = 1$ ,  $h_1 = 3$ , and use it to find a formula for  $h_n$ .
6. Find the generating function for the solutions to  $h_n = 9h_{n-1} - 26h_{n-2} + 24h_{n-3}$ ,  $h_0 = 0$ ,  $h_1 = 1$ ,  $h_2 = -1$ , and use it to find a formula for  $h_n$ .
7. Find the generating function for the solutions to  $h_n = 3h_{n-1} + 4h_{n-2}$ ,  $h_0 = 0$ ,  $h_1 = 1$ , and use it to find a formula for  $h_n$ .
8. Find a recursion for the number of ways to place flags on an  $n$  foot pole, where we have red flags that are 2 feet high, blue flags that are 1 foot high, and yellow flags that are 1 foot high; the heights of the flags must add up to  $n$ . Solve the recursion.
9. In Fibonacci's original problem, a farmer started with one (newborn) pair of rabbits at month 0. After each pair of rabbits was one month old, they produced another pair each month in perpetuity. Thus, after 1 month, he had the original pair, after two months 2 pairs, three months, 3 pairs, four months, 5 pairs, etc. The number of pairs of rabbits satisfies  $h_n = h_{n-1} + h_{n-2}$ ,  $h_0 = h_1 = 1$ . (Note that this is slightly different than our definition, in which  $h_0 = 0$ .)

Suppose instead that each mature pair gives birth to *two* pairs of rabbits. The sequence for the number of pairs of rabbits now starts out  $h_0 = 1$ ,  $h_1 = 1$ ,  $h_2 = 3$ ,  $h_3 = 5$ ,  $h_4 = 11$ . Set up and solve a recurrence relation for the number of pairs of rabbits. Show also that the sequence satisfies  $h_n = 2h_{n-1} + (-1)^n$ .

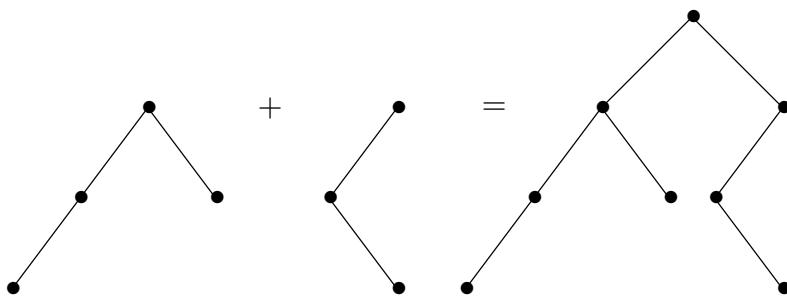
## 3.5 CATALAN NUMBERS

A **rooted binary tree** is a type of graph that is particularly of interest in some areas of computer science. A typical rooted binary tree is shown in figure 3.5.1. The root is the topmost vertex. The vertices below a vertex and connected to it by an edge are the children of the vertex. It is a binary tree because all vertices have 0, 1, or 2 children. How many different rooted binary trees are there with  $n$  vertices?



**Figure 3.5.1** A rooted binary tree.

Let us denote this number by  $C_n$ ; these are the **Catalan numbers**. For convenience, we allow a rooted binary tree to be empty, and let  $C_0 = 1$ . Then it is easy to see that  $C_1 = 1$  and  $C_2 = 2$ , and not hard to see that  $C_3 = 5$ . Notice that any rooted binary tree on at least one vertex can be viewed as two (possibly empty) binary trees joined into a new tree by introducing a new root vertex and making the children of this root the two roots of the original trees; see figure 3.5.2. (To make the empty tree a child of the new vertex, simply do nothing, that is, omit the corresponding child.)



**Figure 3.5.2** Producing a new tree from smaller trees.

Thus, to make all possible binary trees with  $n$  vertices, we start with a root vertex, and then for its two children insert rooted binary trees on  $k$  and  $l$  vertices, with  $k + l = n - 1$ ,

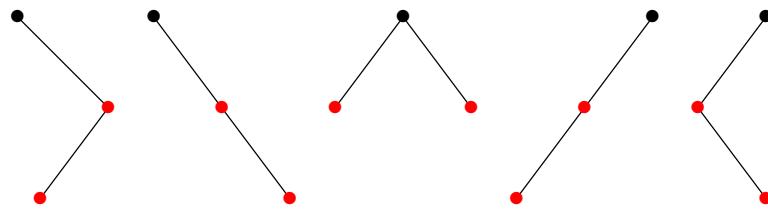
for all possible choices of the smaller trees. Now we can write

$$C_n = \sum_{i=0}^{n-1} C_i C_{n-i-1}.$$

For example, since we know that  $C_0 = C_1 = 1$  and  $C_2 = 2$ ,

$$C_3 = C_0 C_2 + C_1 C_1 + C_2 C_0 = 1 \cdot 2 + 1 \cdot 1 + 2 \cdot 1 = 5,$$

as mentioned above. Once we know the trees on 0, 1, and 2 vertices, we can combine them in all possible ways to list the trees on 3 vertices, as shown in figure 3.5.3. Note that the first two trees have no left child, since the only tree on 0 vertices is empty, and likewise the last two have no right child.



**Figure 3.5.3** The 3-vertex binary rooted trees.

Now we use a generating function to find a formula for  $C_n$ . Let  $f = \sum_{i=0}^{\infty} C_i x^i$ . Now consider  $f^2$ : the coefficient of the term  $x^n$  in the expansion of  $f^2$  is  $\sum_{i=0}^n C_i C_{n-i}$ , corresponding to all possible ways to multiply terms of  $f$  to get an  $x^n$  term:

$$C_0 \cdot C_n x^n + C_1 x \cdot C_{n-1} x^{n-1} + C_2 x^2 \cdot C_{n-2} x^{n-2} + \cdots + C_n x^n \cdot C_0.$$

Now we recognize this as precisely the sum that gives  $C_{n+1}$ , so  $f^2 = \sum_{n=0}^{\infty} C_{n+1} x^n$ . If we multiply this by  $x$  and add 1 (which is  $C_0$ ) we get exactly  $f$  again, that is,  $xf^2 + 1 = f$  or  $xf^2 - f + 1 = 0$ ; here 0 is the zero function, that is,  $xf^2 - f + 1$  is 0 for all  $x$ . Using the Pythagorean theorem,

$$f = \frac{1 \pm \sqrt{1 - 4x}}{2x},$$

as long as  $x \neq 0$ . It is not hard to see that as  $x$  approaches 0,

$$\frac{1 + \sqrt{1 - 4x}}{2x}$$

goes to infinity while

$$\frac{1 - \sqrt{1 - 4x}}{2x}$$

goes to 1. Since we know  $f(0) = C_0 = 1$ , this is the  $f$  we want.

## 68 Chapter 3 Generating Functions

Now by Newton's Binomial Theorem 3.1.1, we can expand

$$\sqrt{1 - 4x} = (1 + (-4x))^{1/2} = \sum_{n=0}^{\infty} \binom{1/2}{n} (-4x)^n.$$

Then

$$\frac{1 - \sqrt{1 - 4x}}{2x} = \sum_{n=1}^{\infty} -\frac{1}{2} \binom{1/2}{n} (-4)^n x^{n-1} = \sum_{n=0}^{\infty} -\frac{1}{2} \binom{1/2}{n+1} (-4)^{n+1} x^n.$$

Expanding the binomial coefficient  $\binom{1/2}{n+1}$  and reorganizing the expression, we discover that

$$C_n = -\frac{1}{2} \binom{1/2}{n+1} (-4)^{n+1} = \frac{1}{n+1} \binom{2n}{n}.$$

In exercise 7 in section 1.2, we saw that the number of properly matched sequences of parentheses of length  $2n$  is  $\binom{2n}{n} - \binom{2n}{n+1}$ , and called this  $C_n$ . It is not difficult to see that

$$\binom{2n}{n} - \binom{2n}{n+1} = \frac{1}{n+1} \binom{2n}{n},$$

so the formulas are in agreement.

Temporarily let  $A_n$  be the number of properly matched sequences of parentheses of length  $2n$ , so from the exercise we know  $A_n = \binom{2n}{n} - \binom{2n}{n+1}$ . It is possible to see directly that  $A_0 = A_1 = 1$  and that the numbers  $A_n$  satisfy the same recurrence relation as do the  $C_n$ , which implies that  $A_n = C_n$ , without manipulating the generating function.

There are many counting problems whose answers turns out to be the Catalan numbers. *Enumerative Combinatorics: Volume 2*, by Richard Stanley, contains a large number of examples.

### Exercises 3.5.

1. Show that

$$\binom{2n}{n} - \binom{2n}{n+1} = \frac{1}{n+1} \binom{2n}{n}.$$

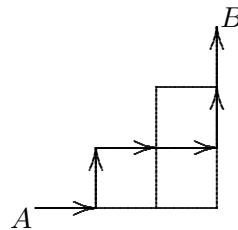
2. Find a simple expression  $f(n)$  so that  $C_{n+1} = f(n)C_n$ . Use this to compute  $C_1, \dots, C_6$  from  $C_0$ .
3. Show that if  $A_n$  is the number of properly matched sequences of parentheses of length  $2n$ , then

$$A_n = \sum_{i=0}^{n-1} A_i A_{n-i-1}.$$

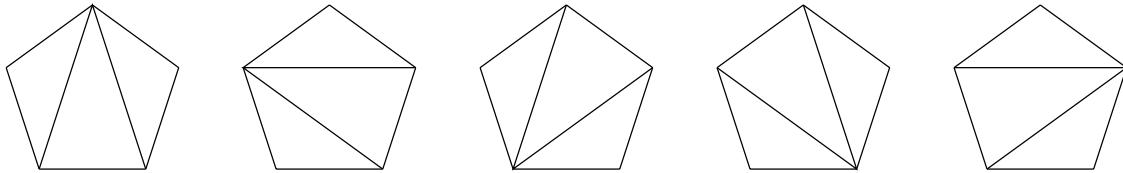
Do this in the same style that we used for the number of rooted binary trees: Given all the sequences of shorter length, explain how to combine them to produce the sequences of length

$2n$ , in such a way that the sum clearly counts the number of sequences. Hint: Prove the following lemma: If  $s$  is a properly matched sequence of parentheses of length  $2n$ ,  $s$  may be written uniquely in the form  $(s_1)s_2$ , where  $s_1$  and  $s_2$  are properly matched sequences of parentheses whose lengths add to  $2n-2$ . For example,  $((())()) = ([()])[()]$  and  $()((()) = ([])[((())]$ , with the sequences  $s_1$  and  $s_2$  indicated by  $[]$ . Note that  $s_1$  and  $s_2$  are allowed to be empty sequences, with length 0.

4. Consider a “staircase” as shown below. A path from  $A$  to  $B$  consists of a sequence of edges starting at  $A$ , ending at  $B$ , and proceeding only up or right; all paths are of length 6. One such path is indicated by arrows. The staircase shown is a “ $3 \times 3$ ” staircase. How many paths are there in an  $n \times n$  staircase?



5. A convex polygon with  $n \geq 3$  sides can be divided into triangles by inserting  $n - 3$  non-intersecting diagonals. In how many different ways can this be done? The possibilities for  $n = 5$  are shown.



6. A **partition** of a set  $S$  is a collection of non-empty subsets  $A_i \subseteq S$ ,  $1 \leq i \leq k$  (the **parts** of the partition), such that  $\bigcup_{i=1}^k A_i = S$  and for every  $i \neq j$ ,  $A_i \cap A_j = \emptyset$ . For example, one partition of  $\{1, 2, 3, 4, 5\}$  is  $\{\{1, 3\}, \{4\}, \{2, 5\}\}$ .

Suppose the integers  $1, 2, \dots, n$  are arranged on a circle, in order around the circle. A partition of  $\{1, 2, \dots, n\}$  is a **non-crossing partition** if it satisfies this additional property: If  $w$  and  $x$  are in some part  $A_i$ , and  $y$  and  $z$  are in a different part  $A_j$ , then the line joining  $w$  to  $x$  does not cross the line joining  $y$  to  $z$ . The partition above,  $\{1, 3\}, \{4\}, \{2, 5\}$ , is not a non-crossing partition, as the line 1–3 crosses the line 2–5.

Find the number of non-crossing partitions of  $\{1, 2, \dots, n\}$ .

Recall from section 1.4 that the Bell numbers count all of the partitions of  $\{1, 2, \dots, n\}$ . Hence, this exercise gives us a lower bound on the total number of partitions.

7. Consider a set of  $2n$  people sitting around a table. In how many ways can we arrange for each person to shake hands with another person at the table such that no two handshakes cross?



# 4

## Systems of Distinct Representatives

Suppose that the student clubs at a college each send a representative to the student government from among the members of the club. No person may represent more than one club; is this possible? It is certainly possible sometimes, for example when no student belongs to two clubs. It is not hard to see that it could be impossible. So the first substantive question is: is there anything useful or interesting we can say about under what conditions it is possible to choose such representatives.

We turn this into a more mathematical situation:

**DEFINITION 4.0.1** Suppose that  $A_1, A_2, \dots, A_n$  are sets, which we refer to as a **set system**. A (complete) **system of distinct representatives** is a set  $\{x_1, x_2, \dots, x_n\}$  such that  $x_i \in A_i$  for all  $i$ , and no two of the  $x_i$  are the same. A (partial) system of distinct representatives is a set of distinct elements  $\{x_1, x_2, \dots, x_k\}$  such that  $x_i \in A_{j_i}$ , where  $j_1, j_2, \dots, j_k$  are distinct integers in  $[n]$ .  $\square$

In standard usage, “system of distinct representatives” means “complete system of distinct representatives”, but it will be convenient to let “system of distinct representatives” mean either a complete or partial system of distinct representatives depending on context. We usually abbreviate “system of distinct representatives” as SDR.

We will analyze this problem in two ways, combinatorially and using graph theory.

## 4.1 EXISTENCE OF SDRs

In this section, SDR means complete SDR. It is easy to see that not every collection of sets has an SDR. For example,

$$A_1 = \{a, b\}, A_2 = \{a, b\}, A_3 = \{a, b\}.$$

The problem is clear: there are only two possible representatives, so a set of three distinct representatives cannot be found. This example is a bit more general than it may at first appear. Consider

$$A_1 = \{a, b\}, A_2 = \{a, b\}, A_3 = \{a, b\}, A_4 = \{b, c, d, e\}.$$

Now the total number of possible representatives is 5, and we only need 4. Nevertheless, this is impossible, because the first three sets have no SDR considered by themselves. Thus the following condition, called **Hall's Condition**, is clearly necessary for the existence of an SDR: For every  $k \geq 1$ , and every set  $\{i_1, i_2, \dots, i_k\} \subseteq [n]$ ,  $|\bigcup_{j=1}^k A_{i_j}| \geq k$ . That is, the number of possible representatives in any collection of sets must be at least as large as the number of sets. Both examples fail to have this property because  $|A_1 \cup A_2 \cup A_3| = 2 < 3$ .

Remarkably, this condition is both necessary and sufficient.

**THEOREM 4.1.1 Hall's Theorem** A collection of sets  $A_1, A_2, \dots, A_n$  has an SDR if and only if for every  $k \geq 1$ , and every set  $\{i_1, i_2, \dots, i_k\} \subseteq [n]$ ,  $|\bigcup_{j=1}^k A_{i_j}| \geq k$ .

**Proof.** We already know the condition is necessary, so we prove sufficiency by induction on  $n$ .

Suppose  $n = 1$ ; the condition is simply that  $|A_1| \geq 1$ . If this is true then  $A_1$  is non-empty and so there is an SDR. This establishes the base case.

Now suppose that the theorem is true for a collection of  $k < n$  sets, and suppose we have sets  $A_1, A_2, \dots, A_n$  satisfying Hall's Condition. We need to show there is an SDR.

Suppose first that for every  $k < n$  and every  $\{i_1, i_2, \dots, i_k\} \subseteq [n]$ , that  $|\bigcup_{j=1}^k A_{i_j}| \geq k + 1$ , that is, that these unions are larger than required. Pick any element  $x_n \in A_n$ , and define  $B_i = A_i \setminus \{x_n\}$  for each  $i < n$ . Consider the collection of sets  $B_1, \dots, B_{n-1}$ , and any union  $\bigcup_{j=1}^k B_{i_j}$  of a subcollection of the sets. There are two possibilities: either  $\bigcup_{j=1}^k B_{i_j} = \bigcup_{j=1}^k A_{i_j}$  or  $\bigcup_{j=1}^k B_{i_j} = \bigcup_{j=1}^k A_{i_j} \setminus \{x_n\}$ , so that  $|\bigcup_{j=1}^k B_{i_j}| = |\bigcup_{j=1}^k A_{i_j}|$  or  $|\bigcup_{j=1}^k B_{i_j}| = |\bigcup_{j=1}^k A_{i_j}| - 1$ . In either case, since  $|\bigcup_{j=1}^k A_{i_j}| \geq k+1$ ,  $|\bigcup_{j=1}^k B_{i_j}| \geq k$ . Thus, by the induction hypothesis, the collection  $B_1, \dots, B_{n-1}$  has an SDR  $\{x_1, x_2, \dots, x_{n-1}\}$ , and for every  $i < n$ ,  $x_i \neq x_n$ , by the definition of the  $B_i$ . Thus  $\{x_1, x_2, \dots, x_n\}$  is an SDR for  $A_1, A_2, \dots, A_n$ .

If it is not true that for every  $k < n$  and every  $\{i_1, i_2, \dots, i_k\} \subseteq [n]$ ,  $|\bigcup_{j=1}^k A_{i_j}| \geq k + 1$ , then for some  $k < n$  and  $\{i_1, i_2, \dots, i_k\}$ ,  $|\bigcup_{j=1}^k A_{i_j}| = k$ . Without loss of generality, we

may assume that  $|\bigcup_{j=1}^k A_j| = k$ . By the induction hypothesis,  $A_1, A_2, \dots, A_k$  has an SDR,  $\{x_1, \dots, x_k\}$ .

Define  $B_i = A_i \setminus \bigcup_{j=1}^k A_j$  for  $i > k$ . Suppose that  $\{x_{k+1}, \dots, x_n\}$  is an SDR for  $B_{k+1}, \dots, B_n$ ; then it is also an SDR for  $A_{k+1}, \dots, A_n$ . Moreover,  $\{x_1, \dots, x_n\}$  is an SDR for  $A_1, \dots, A_n$ . Thus, to finish the proof it suffices to show that  $B_{k+1}, \dots, B_n$  has an SDR. The number of sets here is  $n - k < n$ , so we need only show that the sets satisfy Hall's Condition.

So consider some sets  $B_{i_1}, B_{i_2}, \dots, B_{i_l}$ . First we notice that

$$|A_1 \cup A_2 \cup \dots \cup A_k \cup B_{i_1} \cup B_{i_2} \cup \dots \cup B_{i_l}| = k + |B_{i_1} \cup B_{i_2} \cup \dots \cup B_{i_l}|.$$

Also

$$|A_1 \cup A_2 \cup \dots \cup A_k \cup B_{i_1} \cup B_{i_2} \cup \dots \cup B_{i_l}| = |A_1 \cup A_2 \cup \dots \cup A_k \cup A_{i_1} \cup A_{i_2} \cup \dots \cup A_{i_l}|$$

and

$$|A_1 \cup A_2 \cup \dots \cup A_k \cup A_{i_1} \cup A_{i_2} \cup \dots \cup A_{i_l}| \geq k + l.$$

Putting these together gives

$$k + |B_{i_1} \cup B_{i_2} \cup \dots \cup B_{i_l}| \geq k + l$$

$$|B_{i_1} \cup B_{i_2} \cup \dots \cup B_{i_l}| \geq l$$

Thus,  $B_{k+1}, \dots, B_n$  has an SDR, which finishes the proof. ■

### **Exercises 4.1.**

1. How many different systems of distinct representatives are there for  $A_1 = \{1, 2\}$ ,  $A_2 = \{2, 3\}$ ,  $\dots, A_n = \{n, 1\}$ ?
2. How many different systems of distinct representatives are there for the sets  $A_i = [n] \setminus i$ ,  $i = 1, 2, \dots, n$ ,  $n \geq 2$ ?
3. Suppose the set system  $A_1, A_2, \dots, A_n$  has an SDR, and that  $x \in A_i$ . Show the set system has an SDR containing  $x$ . Show that  $x$  cannot necessarily be chosen to represent  $A_i$ .
4. Suppose the set system  $A_1, A_2, \dots, A_n$  satisfies  $|\bigcup_{j=1}^k A_{i_j}| \geq k + 1$  for every  $1 \leq k < n$  and  $\{i_1, i_2, \dots, i_k\} \subseteq [n]$ , and that  $x \in A_i$ . Show the set system has an SDR in which  $x$  represents  $A_i$ .
5. An  $m \times n$  chessboard, with  $m$  even and both  $m$  and  $n$  at least 2, has one white and one black square removed. Show that the board can be covered by dominoes.

## 4.2 PARTIAL SDRs

In this section, SDR means partial SDR.

If there is no complete SDR, we naturally want to know how many of the  $n$  sets can be represented, that is, what is the largest value of  $m$  so that some  $m$  of the sets have a complete SDR. Since there is no complete SDR, there are sets  $A_{i_1}, A_{i_2}, \dots, A_{i_k}$  such that  $|\bigcup_{j=1}^k A_{i_j}| = l < k$ . Clearly at most  $l$  of these  $k$  sets have a complete SDR, so no SDR for  $A_1, A_2, \dots, A_n$  can be larger than  $n - k + l$ . Thus,  $m$  can be no larger than the minimum value, over all  $k$  and all collections of sets  $A_{i_1}, A_{i_2}, \dots, A_{i_k}$ , of  $n - k + |\bigcup_{j=1}^k A_{i_j}|$ . Note that if  $|\bigcup_{j=1}^k A_{i_j}| > k$ ,  $n - k + |\bigcup_{j=1}^k A_{i_j}| > n$ , which tells us nothing. If  $k = 0$ ,  $n - k + |\bigcup_{j=1}^k A_{i_j}| = n$  (because empty unions are empty), so we are guaranteed that the minimum is never greater than  $n$ . In fact the minimum value of the expression is exactly the size of a largest SDR.

**THEOREM 4.2.1** The maximum size of an SDR for the sets  $A_1, A_2, \dots, A_n$  is the minimum value, for  $0 \leq k \leq n$  and sets  $A_{i_1}, A_{i_2}, \dots, A_{i_k}$ , of  $n - k + |\bigcup_{j=1}^k A_{i_j}|$ .

**Proof.** Since no SDR can be larger than this minimum value, it suffices to show that we can find an SDR whose size is this minimum. The proof is by induction on  $n$ ; the case  $n = 1$  is easy.

Suppose first that the minimum value is  $n$ , so that for all  $k$  and all collections of sets  $A_{i_1}, A_{i_2}, \dots, A_{i_k}$ ,

$$n - k + \left| \bigcup_{j=1}^k A_{i_j} \right| \geq n.$$

Then rearranging we see that

$$\left| \bigcup_{j=1}^k A_{i_j} \right| \geq k,$$

so by Hall's Theorem (4.1.1), there is an SDR of size  $n$ .

Note that the minimum value of  $n - k + |\bigcup_{j=1}^k A_{i_j}|$  occurs when  $|\bigcup_{j=1}^k A_{i_j}| - k$  is a minimum, that is

$$\min(n - k + \left| \bigcup_{j=1}^k A_{i_j} \right|) = n + \min(\left| \bigcup_{j=1}^k A_{i_j} \right| - k).$$

Suppose now that the minimum  $m$  is less than  $n$ , and that  $m = n - k + |\bigcup_{j=1}^k A_{i_j}|$ , with  $0 < k < n$ . Let  $B_j = A_{i_j}$ ; since  $k < n$ , the induction hypothesis applies to the sets  $B_1, \dots, B_k$ . Since each set  $B_j$  is  $A_{i_j}$ ,  $\left| \bigcup_{j=1}^l B_{h_j} \right| - l \geq \left| \bigcup_{j=1}^k A_{i_j} \right| - k$ , for all  $l$  and  $B_{h_1}, \dots, B_{h_l}$ . Thus, the minimum value of  $\left| \bigcup_{j=1}^l B_{h_j} \right| - l$ , over all  $l$  and  $B_{h_1}, \dots, B_{h_l}$ , is

$|\bigcup_{j=1}^k B_j| - k = |\bigcup_{j=1}^k A_{i_j}| - k$ , so by the induction hypothesis, the sets  $A_{i_1}, A_{i_2}, \dots, A_{i_k}$  have an SDR of size  $k - k + |\bigcup_{j=1}^k A_{i_j}| = |\bigcup_{j=1}^k A_{i_j}| = m - n + k$ ,  $\{x_1, \dots, x_{m-n+k}\}$ .

Now consider the  $n - k$  sets consisting of those original sets not in  $A_{i_1}, A_{i_2}, \dots, A_{i_k}$ , that is,  $\{A_i \mid i \notin \{i_1, \dots, i_k\}\}$ . Let  $C_i = A_i \setminus \bigcup_{j=1}^k A_{i_j}$  for  $i$  not in  $i_1, i_2, \dots, i_k$ . Consider some sets  $C_{g_1}, C_{g_2}, \dots, C_{g_l}$ . If  $|\bigcup_{j=1}^l C_{g_j}| < l$  then  $|\bigcup_{j=1}^l C_{g_j}| - l < 0$  and

$$\begin{aligned} n - k + \left| \bigcup_{j=1}^k A_{i_j} \right| &> n - k - l + \left| \bigcup_{j=1}^l C_{g_j} \right| + \left| \bigcup_{j=1}^k A_{i_j} \right| \\ &\geq n - (k + l) + |C_{g_1} \cup \dots \cup C_{g_l} \cup A_{i_1} \cup \dots \cup A_{i_k}| \\ &= n - (k + l) + |A_{g_1} \cup \dots \cup A_{g_l} \cup A_{i_1} \cup \dots \cup A_{i_k}|, \end{aligned}$$

contradicting the fact that  $n - k + |\bigcup_{j=1}^k A_{i_j}|$  is a minimum. Thus by Hall's Theorem (4.1.1), the sets  $C_{g_1}, C_{g_2}, \dots, C_{g_{n-k}}$  have a complete SDR  $\{y_1, \dots, y_{n-k}\}$ . By the definition of the sets  $C_i$ ,  $\{x_1, \dots, x_{m-n+k}\} \cap \{y_1, \dots, y_{n-k}\} = \emptyset$ , so  $\{x_1, \dots, x_{m-n+k}\} \cup \{y_1, \dots, y_{n-k}\}$  is an SDR of size  $m - n + k + n - k = m$  as desired.

Finally, suppose that the minimum value of  $n - k + |\bigcup_{j=1}^k A_{i_j}|$  occurs only when  $k = n$ , so we want an SDR of size

$$n - n + \left| \bigcup_{j=1}^n A_j \right| = \left| \bigcup_{j=1}^n A_j \right|.$$

Then

$$\begin{aligned} n - (n - 1) + \left| \bigcup_{j=1}^{n-1} A_j \right| &> \left| \bigcup_{j=1}^n A_j \right| \\ 1 + \left| \bigcup_{j=1}^{n-1} A_j \right| &> \left| \bigcup_{j=1}^n A_j \right| \\ \left| \bigcup_{j=1}^{n-1} A_j \right| &\geq \left| \bigcup_{j=1}^n A_j \right|. \end{aligned}$$

Since  $|\bigcup_{j=1}^{n-1} A_j| \leq |\bigcup_{j=1}^n A_j|$ ,  $|\bigcup_{j=1}^{n-1} A_j| = |\bigcup_{j=1}^n A_j|$ . By the induction hypothesis, the theorem applies to the sets  $A_1, A_2, \dots, A_{n-1}$ . If the minimum of  $(n - 1) - l + |\bigcup_{j=1}^l A_{i_j}|$  occurs when  $l = n - 1$ , then there is an SDR of size  $(n - 1) - (n - 1) + |\bigcup_{j=1}^{n-1} A_j| = |\bigcup_{j=1}^{n-1} A_j| = |\bigcup_{j=1}^n A_j|$ , as desired.

If the minimum occurs when  $l < n - 1$  and not when  $l = n - 1$ , then

$$\begin{aligned} (n - 1) - l + \left| \bigcup_{j=1}^l A_{i_j} \right| &< \left| \bigcup_{j=1}^{n-1} A_j \right| \\ n - l + \left| \bigcup_{j=1}^l A_{i_j} \right| &< \left| \bigcup_{j=1}^{n-1} A_j \right| + 1 \end{aligned}$$

and by assumption

$$n - l + \left| \bigcup_{j=1}^l A_{i_j} \right| > \left| \bigcup_{j=1}^n A_j \right|.$$

Thus

$$\begin{aligned} \left| \bigcup_{j=1}^n A_j \right| &< n - l + \left| \bigcup_{j=1}^l A_{i_j} \right| \\ &< \left| \bigcup_{j=1}^{n-1} A_j \right| + 1 \\ &= \left| \bigcup_{j=1}^n A_j \right| + 1. \end{aligned}$$

This means that there is an integer strictly between two consecutive integers, a contradiction. This completes the proof. ■

While this theorem provides a method to calculate the size of a maximum SDR, the method is hardly efficient: it requires looking at all possible collections of the sets. It also does not provide a way to find an actual SDR, that is, the actual representatives. We will fix these problems in the last two sections of this chapter.

### *Exercises 4.2.*

- Find the size of a maximum SDR for

$$A_1 = \{a, b, c\}, A_2 = \{a, b, c, d, e\}, A_3 = \{a, b\}, A_4 = \{b, c\}, A_5 = \{a\}, A_6 = \{a, c, e\}.$$

Justify your answer.

## 4.3 LATIN SQUARES

**DEFINITION 4.3.1** A **Latin square** of order  $n$  is an  $n \times n$  grid filled with  $n$  symbols so that each symbol appears once in each row and column. □

**EXAMPLE 4.3.2** Here is a Latin square of order 4:

♥	♣	♠	◊
♣	♠	◊	♥
♠	◊	♥	♣
◊	♥	♣	♠

□

Usually we use the integers  $1 \dots n$  for the symbols. There are many, many Latin squares of order  $n$ , so it pays to limit the number by agreeing not to count Latin squares

that are “really the same” as different. The simplest way to do this is to consider **reduced** Latin squares. A reduced Latin square is one in which the first row is  $1 \dots n$  (in order) and the first column is likewise  $1 \dots n$ .

**EXAMPLE 4.3.3** Consider this Latin square:

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

The order of the rows and columns is not really important to the idea of a Latin square. If we reorder the rows and columns, we can consider the result to be in essence the same Latin square. By reordering the columns, we can turn the square above into this:

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

Then we can swap rows two and three:

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

This Latin square is in reduced form, and is essentially the same as the original.  $\square$

Another simple way to change the appearance of a Latin square without changing its essential structure is to interchange the symbols.

**EXAMPLE 4.3.4** Starting with the same Latin square as before:

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

we can interchange the symbols 1 and 4 to get:

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

Now if we swap rows three and four we get:

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

Notice that this Latin square is in reduced form, but it is not the same as the reduced form from the previous example, even though we started with the same Latin square. Thus, we may want to consider some reduced Latin squares to be the same as each other.  $\square$

**DEFINITION 4.3.5** Two Latin squares are **isotopic** if each can be turned into the other by permuting the rows, columns, and symbols. This isotopy relation is an equivalence relation; the equivalence classes are the **isotopy classes**.  $\square$

Latin squares are apparently quite difficult to count without substantial computing power. According to [Wikipedia](#), the number of Latin squares is known only up to  $n = 11$ . Here are the first few values for all Latin squares, reduced Latin squares, and non-isotopic Latin squares (that is, the number of isotopy classes):

$n$	All	Reduced	Non-isotopic
1	1	1	1
2	2	1	1
3	12	1	1
4	576	4	2
5	161280	56	2

How can we produce a Latin square? If you know what a group is, you should know that the multiplication table of any finite group is a Latin square. (Also, any Latin square is the multiplication table of a **quasigroup**.) Even if you have not encountered groups by that name, you may know of some. For example, considering the integers modulo  $n$  under addition, the addition table is a Latin square.

**EXAMPLE 4.3.6** Here is the addition table for the integers modulo 6:

0	1	2	3	4	5
1	2	3	4	5	0
2	3	4	5	0	1
3	4	5	0	1	2
4	5	0	1	2	3
5	0	1	2	3	4

□

**EXAMPLE 4.3.7** Here is another way to potentially generate many Latin squares. Start with first row  $1, \dots, n$ . Consider the sets  $A_i = [n] \setminus \{i\}$ . From exercise 1 in section 4.1 we know that this set system has many SDRs; if  $x_1, x_2, \dots, x_n$  is an SDR, we may use it for row two. In general, after we have chosen rows  $1, \dots, j$ , we let  $A_i$  be the set of integers that have not yet been chosen for column  $i$ . This set system has an SDR, which we use for row  $j + 1$ . □

**DEFINITION 4.3.8** Suppose  $A$  and  $B$  are two Latin squares of order  $n$ , with entries  $A_{i,j}$  and  $B_{i,j}$  in row  $i$  and column  $j$ . Form the matrix  $M$  with entries  $M_{i,j} = (A_{i,j}, B_{i,j})$ ; we will denote this operation as  $M = A \cup B$ . We say that  $A$  and  $B$  are **orthogonal** if  $M$  contains all  $n^2$  ordered pairs  $(a, b)$ ,  $1 \leq a \leq n$ ,  $1 \leq b \leq n$ , that is, all elements of  $\{0, 1, \dots, n-1\} \times \{0, 1, \dots, n-1\}$ . □

As we will see, it is easy to find orthogonal Latin squares of order  $n$  if  $n$  is odd; not too hard to find orthogonal Latin squares of order  $4k$ , and difficult but possible to find orthogonal Latin squares of order  $4k+2$ , with the exception of orders 2 and 6. In the 1700s, Euler showed that there are orthogonal Latin squares of all orders except of order  $4k+2$ , and he conjectured that there are no orthogonal Latin squares of order 6. In 1901, the amateur mathematician Gaston Tarry showed that indeed there are none of order 6, by showing that all possibilities for such Latin squares failed to be orthogonal. In 1959 it was finally shown that there are orthogonal Latin squares of all other orders.

**THEOREM 4.3.9** There are pairs of orthogonal Latin squares of order  $n$  when  $n$  is odd.

**Proof.** This proof can be shortened by using ideas of group theory, but we will present a self-contained version. Consider the addition table for addition mod  $n$ :

	0	$\cdots$	$j$	$\cdots$	$n - 1$
0	0	$\cdots$	$j$	$\cdots$	$n - 1$
$\vdots$					
$i$	$i$	$\cdots$	$i + j$	$\cdots$	$n + i - 1$
$\vdots$					
$n - 1$	$n - 1$	$\cdots$	$n + j - 1$	$\cdots$	$n - 2$

We claim first that this (without the first row and column, of course) is a Latin square with symbols  $0, 1, \dots, n - 1$ . Consider two entries in row  $i$ , say  $i + j$  and  $i + k$ . If  $i + j \equiv i + k \pmod{n}$ , then  $j \equiv k$ , so  $j = k$ . Thus, all entries of row  $i$  are distinct, so each of  $0, 1, \dots, n - 1$  appears exactly once in row  $i$ . The proof that each appears once in any column is similar. Call this Latin square  $A$ . (Note that so far everything is true whether  $n$  is odd or even.)

Now form a new square  $B$  with entries  $B_{i,j} = A_{2i,j} = 2i + j$ , where by  $2i$  and  $2i + j$  we mean those values mod  $n$ . Thus row  $i$  of  $B$  is the same as row  $2i$  of  $A$ . Now we claim that in fact the rows of  $B$  are exactly the rows of  $A$ , in a different order. To do this, it suffices to show that if  $2i \equiv 2k \pmod{n}$ , then  $i = k$ . This implies that all the rows of  $B$  are distinct, and hence must be all the rows of  $A$ .

Suppose without loss of generality that  $i \geq k$ . If  $2i \equiv 2k \pmod{n}$  then  $n \mid 2(i - k)$ . Since  $n$  is odd,  $n \mid (i - k)$ . Since  $i$  and  $k$  are in  $0, 1, \dots, n - 1$ ,  $0 \leq i - k \leq n - 1$ . Of these values, only 0 is divisible by  $n$ , so  $i - k = 0$ . Thus  $B$  is also a Latin square.

To show that  $A \cup B$  contains all  $n^2$  elements of  $\{0, 1, \dots, n - 1\} \times \{0, 1, \dots, n - 1\}$ , it suffices to show that no two elements of  $A \cup B$  are the same. Suppose that  $(i_1 + j_1, 2i_1 + j_1) = (i_2 + j_2, 2i_2 + j_2)$  (arithmetic is mod  $n$ ). Then by subtracting equations,  $i_1 = i_2$ ; with the first equation this implies  $j_1 = j_2$ . ■

**EXAMPLE 4.3.10** When  $n = 3$ ,

$$\begin{bmatrix} 0 & 1 & 2 \\ 1 & 2 & 0 \\ 2 & 0 & 1 \end{bmatrix} \cup \begin{bmatrix} 0 & 1 & 2 \\ 2 & 0 & 1 \\ 1 & 2 & 0 \end{bmatrix} = \begin{bmatrix} (0,0) & (1,1) & (2,2) \\ (1,2) & (2,0) & (0,1) \\ (2,1) & (0,2) & (1,0) \end{bmatrix}.$$

□

One obvious approach to constructing Latin squares, and pairs of orthogonal Latin squares, is to start with smaller Latin squares and use them to produce larger ones. We will produce a Latin square of order  $mn$  from a Latin square of order  $m$  and one of order  $n$ .

Let  $A$  be a Latin square of order  $m$  with symbols  $1, \dots, m$ , and  $B$  one of order  $n$  with symbols  $1, \dots, n$ . Let  $c_{i,j}$ ,  $1 \leq i \leq m$ ,  $1 \leq j \leq n$ , be  $mn$  new symbols. Form an  $mn \times mn$

grid by replacing each entry of  $B$  with a copy of  $A$ . Then replace each entry  $i$  in this copy of  $A$  with  $c_{i,j}$ , where  $j$  is the entry of  $B$  that was replaced. We denote this new Latin square  $A \times B$ . Here is an example, combining a  $4 \times 4$  Latin square with a  $3 \times 3$  Latin square to form a  $12 \times 12$  Latin square:

$$\begin{array}{|c|c|c|c|} \hline 1 & 2 & 3 & 4 \\ \hline 2 & 3 & 4 & 1 \\ \hline 3 & 4 & 1 & 2 \\ \hline 4 & 1 & 2 & 3 \\ \hline \end{array}
 \times
 \begin{array}{|c|c|c|} \hline 1 & 2 & 3 \\ \hline 2 & 3 & 1 \\ \hline 3 & 1 & 2 \\ \hline \end{array}
 =
 \begin{array}{|c|c|c|c|c|c|c|c|c|c|c|c|} \hline c_{1,1} & c_{2,1} & c_{3,1} & c_{4,1} & c_{1,2} & c_{2,2} & c_{3,2} & c_{4,2} & c_{1,3} & c_{2,3} & c_{3,3} & c_{4,3} \\ \hline c_{2,1} & c_{3,1} & c_{4,1} & c_{1,1} & c_{2,2} & c_{3,2} & c_{4,2} & c_{1,2} & c_{2,3} & c_{3,3} & c_{4,3} & c_{1,3} \\ \hline c_{3,1} & c_{4,1} & c_{1,1} & c_{2,1} & c_{3,2} & c_{4,2} & c_{1,2} & c_{2,2} & c_{3,3} & c_{4,3} & c_{1,3} & c_{2,3} \\ \hline c_{4,1} & c_{1,1} & c_{2,1} & c_{3,1} & c_{4,2} & c_{1,2} & c_{2,2} & c_{3,2} & c_{4,3} & c_{1,3} & c_{2,3} & c_{3,3} \\ \hline c_{1,2} & c_{2,2} & c_{3,2} & c_{4,2} & c_{1,3} & c_{2,3} & c_{3,3} & c_{4,3} & c_{1,1} & c_{2,1} & c_{3,1} & c_{4,1} \\ \hline c_{2,2} & c_{3,2} & c_{4,2} & c_{1,2} & c_{2,3} & c_{3,3} & c_{4,3} & c_{1,3} & c_{2,1} & c_{3,1} & c_{4,1} & c_{1,1} \\ \hline c_{3,2} & c_{4,2} & c_{1,2} & c_{2,2} & c_{3,3} & c_{4,3} & c_{1,3} & c_{2,3} & c_{3,1} & c_{4,1} & c_{1,1} & c_{2,1} \\ \hline c_{4,2} & c_{1,2} & c_{2,2} & c_{3,2} & c_{4,3} & c_{1,3} & c_{2,3} & c_{3,3} & c_{4,1} & c_{1,1} & c_{2,1} & c_{3,1} \\ \hline c_{1,3} & c_{2,3} & c_{3,3} & c_{4,3} & c_{1,1} & c_{2,1} & c_{3,1} & c_{4,1} & c_{1,2} & c_{2,2} & c_{3,2} & c_{4,2} \\ \hline c_{2,3} & c_{3,3} & c_{4,3} & c_{1,3} & c_{2,1} & c_{3,1} & c_{4,1} & c_{1,1} & c_{2,2} & c_{3,2} & c_{4,2} & c_{1,2} \\ \hline c_{3,3} & c_{4,3} & c_{1,3} & c_{2,3} & c_{3,1} & c_{4,1} & c_{1,1} & c_{2,1} & c_{3,2} & c_{4,2} & c_{1,2} & c_{2,2} \\ \hline c_{4,3} & c_{1,3} & c_{2,3} & c_{3,3} & c_{4,1} & c_{1,1} & c_{2,1} & c_{3,1} & c_{4,2} & c_{1,2} & c_{2,2} & c_{3,2} \\ \hline \end{array}$$

**THEOREM 4.3.11** If  $A$  and  $B$  are Latin squares, so is  $A \times B$ .

**Proof.** Consider two symbols  $c_{i,j}$  and  $c_{k,l}$  in the same row. If the positions containing these symbols are in the same copy of  $A$ , then  $i \neq k$ , since  $A$  is a Latin square, and so the symbols  $c_{i,j}$  and  $c_{k,l}$  are distinct. Otherwise,  $j \neq l$ , since  $B$  is a Latin square. The argument is the same for columns. ■

Remarkably, this operation preserves orthogonality:

**THEOREM 4.3.12** If  $A_1$  and  $A_2$  are Latin squares of order  $m$ ,  $B_1$  and  $B_2$  are Latin squares of order  $n$ ,  $A_1$  and  $A_2$  are orthogonal, and  $B_1$  and  $B_2$  are orthogonal, then  $A_1 \times B_1$  is orthogonal to  $A_2 \times B_2$ .

**Proof.** We denote the contents of  $A_i \times B_i$  by  $C_i(w, x, y, z)$ , meaning the entry in row  $w$  and column  $x$  of the copy of  $A_i$  that replaced the entry in row  $y$  and column  $z$  of  $B_i$ , which we denote  $B_i(y, z)$ . We use  $A_i(w, x)$  to denote the entry in row  $w$  and column  $x$  of  $A_i$ .

Suppose that  $(C_1(w, x, y, z), C_2(w, x, y, z)) = (C_1(w', x', y', z'), C_2(w', x', y', z'))$ , where  $(w, x, y, z) \neq (w', x', y', z')$ . Either  $(w, x) \neq (w', x')$  or  $(y, z) \neq (y', z')$ . If the latter, then  $(B_1(y, z), B_2(y, z)) = (B_1(y', z'), B_2(y', z'))$ , a contradiction, since  $B_1$  is orthogonal to  $B_2$ . Hence  $(y, z) = (y', z')$  and  $(w, x) \neq (w', x')$ . But this implies that

## 82 Chapter 4 Systems of Distinct Representatives

$(A_1(w, x), A_2(w, x)) = (A_1(w', x'), A_2(w', x'))$ , a contradiction. Hence  $A_1 \times B_1$  is orthogonal to  $A_1 \times B_2$ . ■

We want to construct orthogonal Latin squares of order  $4k$ . Write  $4k = 2^m \cdot n$ , where  $n$  is odd and  $m \geq 2$ . We know there are orthogonal Latin squares of order  $n$ , by theorem 4.3.9. If there are orthogonal Latin squares of order  $2^m$ , then by theorem 4.3.12 we can construct orthogonal Latin squares of order  $4k = 2^m \cdot n$ .

To get a Latin square of order  $2^m$ , we also use theorem 4.3.12. It suffices to find two orthogonal Latin squares of order  $4 = 2^2$  and two of order  $8 = 2^3$ . Then repeated application of theorem 4.3.12 allows us to build orthogonal Latin squares of order  $2^m$ ,  $m \geq 2$ .

Two orthogonal Latin squares of order 4:

$$\begin{bmatrix} 1 & 2 & 3 & 4 \\ 2 & 1 & 4 & 3 \\ 3 & 4 & 1 & 2 \\ 4 & 3 & 2 & 1 \end{bmatrix} \begin{bmatrix} 1 & 2 & 3 & 4 \\ 3 & 4 & 1 & 2 \\ 4 & 3 & 2 & 1 \\ 2 & 1 & 4 & 3 \end{bmatrix},$$

and two of order 8:

$$\begin{bmatrix} 1 & 3 & 4 & 5 & 6 & 7 & 8 & 2 \\ 5 & 2 & 7 & 1 & 8 & 4 & 6 & 3 \\ 6 & 4 & 3 & 8 & 1 & 2 & 5 & 7 \\ 7 & 8 & 5 & 4 & 2 & 1 & 3 & 6 \\ 8 & 7 & 2 & 6 & 5 & 3 & 1 & 4 \\ 2 & 5 & 8 & 3 & 7 & 6 & 4 & 1 \\ 3 & 1 & 6 & 2 & 4 & 8 & 7 & 5 \\ 4 & 6 & 1 & 7 & 3 & 5 & 2 & 8 \end{bmatrix} \begin{bmatrix} 1 & 4 & 5 & 6 & 7 & 8 & 2 & 3 \\ 8 & 2 & 6 & 5 & 3 & 1 & 4 & 7 \\ 2 & 8 & 3 & 7 & 6 & 4 & 1 & 5 \\ 3 & 6 & 2 & 4 & 8 & 7 & 5 & 1 \\ 4 & 1 & 7 & 3 & 5 & 2 & 8 & 6 \\ 5 & 7 & 1 & 8 & 4 & 6 & 3 & 2 \\ 6 & 3 & 8 & 1 & 2 & 5 & 7 & 4 \\ 7 & 5 & 4 & 2 & 1 & 3 & 6 & 8 \end{bmatrix}.$$

### Exercises 4.3.

1. Show that there is only one reduced Latin square of order 3.
2. Verify that the isotopy relation is an equivalence relation.
3. Find all 4 reduced Latin squares of order 4. Show that there are at most 2 isotopy classes for order 4.
4. Show that the second set system defined in example 4.3.7 has an SDR as claimed.
5. Show that there are no orthogonal Latin squares of order 2.
6. Find the two orthogonal Latin squares of order 5 as described in theorem 4.3.9. Show your answer as in example 4.3.10.
7. Prove that to construct orthogonal Latin squares of order  $2^m$ ,  $m \geq 2$ , it suffices to find two orthogonal Latin squares of order  $4 = 2^2$  and two of order  $8 = 2^3$ .
8. An  $n \times n$  Latin square  $A$  is **symmetric** if it is symmetric around the main diagonal, that is,  $A_{i,j} = A_{j,i}$  for all  $i$  and  $j$ . It is easy to find symmetric Latin squares: every addition table

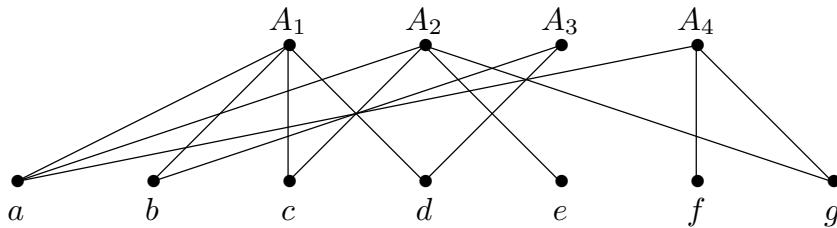
modulo  $n$  is an example, as in example 4.3.6. A Latin square is **idempotent** if every symbol appears on the main diagonal. Show that if  $A$  is both symmetric and idempotent, then  $n$  is odd. Find a  $5 \times 5$  symmetric, idempotent Latin square.

9. The **transpose**  $A^\top$  of a Latin square  $A$  is the reflection of  $A$  across the main diagonal, so that  $A_{i,j}^\top = A_{j,i}$ . A Latin square is self-orthogonal if  $A$  is orthogonal to  $A^\top$ . Show that there is no self-orthogonal Latin square of order 3. Find one of order 4.

## 4.4 INTRODUCTION TO GRAPH THEORY

We can interpret the SDR problem as a problem about graphs. Given sets  $A_1, A_2, \dots, A_n$ , with  $\bigcup_{i=1}^n A_i = \{x_1, x_2, \dots, x_m\}$ , we define a graph with  $n + m$  vertices as follows: The vertices are labeled  $\{A_1, A_2, \dots, A_n, x_1, x_2, \dots, x_m\}$ , and the edges are  $\{\{A_i, x_j\} \mid x_j \in A_i\}$ .

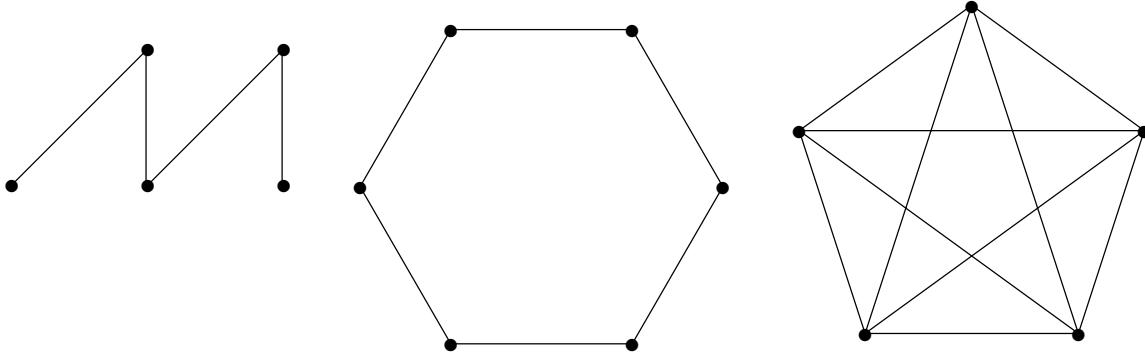
**EXAMPLE 4.4.1** Let  $A_1 = \{a, b, c, d\}$ ,  $A_2 = \{a, c, e, g\}$ ,  $A_3 = \{b, d\}$ , and  $A_4 = \{a, f, g\}$ . The corresponding graph is shown in figure 4.4.1.  $\square$



**Figure 4.4.1** A set system depicted as a bipartite graph.

Before exploring this idea, we introduce a few basic concepts about graphs. If two vertices in a graph are connected by an edge, we say the vertices are **adjacent**. If a vertex  $v$  is an endpoint of edge  $e$ , we say they are **incident**. The set of vertices adjacent to  $v$  is called the **neighborhood** of  $v$ , denoted  $N(v)$ . This is sometimes called the **open neighborhood** of  $v$  to distinguish it from the **closed neighborhood** of  $v$ ,  $N[v] = N(v) \cup \{v\}$ . The **degree** of a vertex  $v$  is the number of edges incident with  $v$ ; it is denoted  $d(v)$ .

Some simple types of graph come up often: A **path** is a graph  $P_n$  on vertices  $v_1, v_2, \dots, v_n$ , with edges  $\{v_i, v_{i+1}\}$  for  $1 \leq i \leq n - 1$ , and no other edges. A **cycle** is a graph  $C_n$  on vertices  $v_1, v_2, \dots, v_n$  with edges  $\{v_i, v_{1+(i \bmod n)}\}$  for  $1 \leq i \leq n$ , and no other edges; this is a path in which the first and last vertices have been joined by an edge. (Generally, we require that a cycle have at least three vertices. If it has two, then the two are joined by two distinct edges; when a graph has more than one edge with the same endpoints it is called a **multigraph**. If a cycle has one vertex, there is an edge, called a **loop**, in which a single vertex serves as both endpoints.) The **length** of a path or cycle is the number of edges in the graph. For example,  $P_1$  has length 0,  $C_1$  has length 1. A **complete graph**

**Figure 4.4.2** Graphs  $P_5$ ,  $C_6$ ,  $K_5$ .

$K_n$  is a graph on  $v_1, v_2, \dots, v_n$  in which every two distinct vertices are joined by an edge. See figure 4.4.2 for examples.

The graph in figure 4.4.1 is a **bipartite graph**.

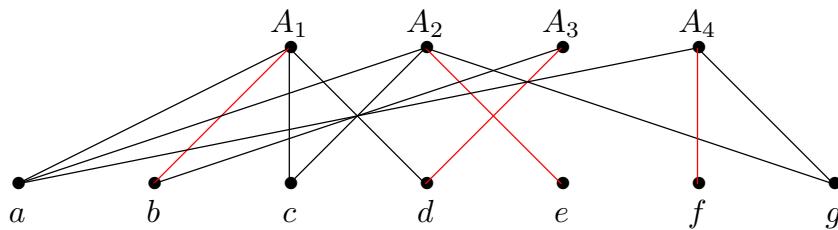
**DEFINITION 4.4.2** A graph  $G$  is bipartite if its vertices can be partitioned into two parts, say  $\{v_1, v_2, \dots, v_n\}$  and  $\{w_1, w_2, \dots, w_m\}$  so that all edges join some  $v_i$  to some  $w_j$ ; no two vertices  $v_i$  and  $v_j$  are adjacent, nor are any vertices  $w_i$  and  $w_j$ .  $\square$

The graph in figure 4.4.1 is bipartite, as are the first two graphs in figure 4.4.2.

## 4.5 MATCHINGS

Now we return to systems of distinct representatives.

A system of distinct representatives corresponds to a set of edges in the corresponding bipartite graph that share no endpoints; such a collection of edges (in any graph, not just a bipartite graph) is called a **matching**. In figure 4.5.1, a matching is shown in red. This is a largest possible matching, since it contains edges incident with all four of the top vertices, and it thus corresponds to a complete SDR.

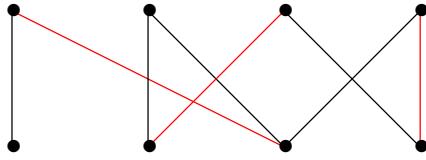
**Figure 4.5.1** A set system depicted as a bipartite graph.

Any bipartite graph can be interpreted as a set system: we simply label all the vertices in one part with “set names”  $A_1, A_2$ , etc., and the other part is labeled with “element

names”, and the sets are defined in the obvious way:  $A_i$  is the neighborhood of the vertex labeled “ $A_i$ ”. Thus, we know one way to compute the size of a maximum matching, namely, we interpret the bipartite graph as a set system and compute the size of a maximum SDR; this is the size of a maximum matching.

We will see another way to do this working directly with the graph. There are two advantages to this: it will turn out to be more efficient, and as a by-product it will actually find a maximum matching.

Given a bipartite graph, it is easy to find a *maximal* matching, that is, one that cannot be made larger simply by adding an edge: just choose edges that do not share endpoints until this is no longer possible. See figure 4.5.2 for an example.

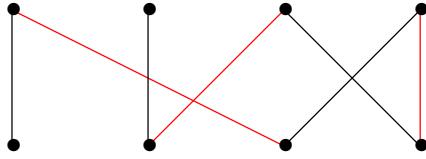


**Figure 4.5.2** A maximal matching is shown in red.

An obvious approach is then to attempt to make the matching larger. There is a simple way to do this, if it works: We look for an **alternating chain**, defined as follows.

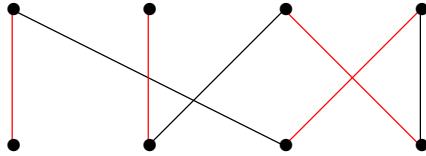
**DEFINITION 4.5.1** Suppose  $M$  is a matching, and suppose that  $v_1, w_1, v_2, w_2, \dots, v_k, w_k$  is a sequence of vertices such that no edge in  $M$  is incident with  $v_1$  or  $w_k$ , and moreover for all  $1 \leq i \leq k$ ,  $v_i$  and  $w_i$  are joined by an edge not in  $M$ , and for all  $1 \leq i \leq k - 1$ ,  $w_i$  and  $v_{i+1}$  are joined by an edge in  $M$ . Then the sequence of vertices together with the edges joining them in order is an alternating chain.  $\square$

The graph in figure 4.5.2 contains alternating chains, one of which is shown in figure 4.5.3.



**Figure 4.5.3** An alternating chain.

Suppose now that we remove from  $M$  all the edges that are in the alternating chain and also in  $M$ , forming  $M'$ , and add to  $M'$  all of the edges in the alternating chain not in  $M$ , forming  $M''$ . It is not hard to show that  $M''$  is a matching, and it contains one more edge than  $M$ . See figure 4.5.4.



**Figure 4.5.4** A new, larger matching.

Remarkably, if there is no alternating chain, then the matching  $M$  is a maximum matching.

**THEOREM 4.5.2** Suppose that  $M$  is a matching in a bipartite graph  $G$ , and there is no alternating chain. Then  $M$  is a maximum matching.

**Proof.** We prove the contrapositive: Suppose that  $M$  is not a maximum matching. Then there is a larger matching,  $N$ . Create a new graph  $G'$  by eliminating all edges that are in both  $M$  and  $N$ , and also all edges that are in neither. We are left with just those edges in  $M$  or  $N$  but not both.

In this new graph no vertex is incident with more than two edges, since if  $v$  were incident with three edges, at least two of them would belong to  $M$  or two would belong to  $N$ , but that can't be true since  $M$  and  $N$  are matchings. This means that  $G'$  is composed of disjoint paths and cycles. Since  $N$  is larger than  $M$ ,  $G'$  contains more edges from  $N$  than from  $M$ , and therefore one of the paths starts and ends with an edge from  $N$ , and along the path the edges alternate between edges in  $N$  and edges in  $M$ . In the original graph  $G$  with matching  $M$ , this path forms an alternating chain. The “alternating” part is clear; we need to see that the first and last vertices in the path are not incident with any edge in  $M$ .

Suppose that the first two vertices are  $v_1$  and  $v_2$ . Then  $v_1$  and  $v_2$  are joined by an edge of  $N$ . Suppose that  $v_1$  is adjacent to a vertex  $w$  and that the edge between  $v_1$  and  $w$  is in  $M$ . This edge cannot be in  $N$ , for then there would be two edges of  $N$  incident at  $v_1$ . But then this edge is in  $G'$ , since it is in  $M$  but not  $N$ , and therefore the path in  $G'$  does not start with the edge in  $N$  joining  $v_1$  and  $v_2$ . This contradiction shows that no edge of  $M$  is incident at  $v_1$ . The proof that the last vertex in the path is likewise not incident with an edge of  $M$  is essentially identical. ■

Now to find a maximum matching, we repeatedly look for alternating chains; when we cannot find one, we know we have a maximum matching. What we need now is an efficient algorithm for finding the alternating chain.

The key, in a sense, is to look for all possible alternating chains simultaneously. Suppose we have a bipartite graph with vertex partition  $\{v_1, v_2, \dots, v_n\}$  and  $\{w_1, w_2, \dots, w_m\}$

and a matching  $M$ . The algorithm labels vertices in such a way that if it succeeds, the alternating chain is indicated by the labels. Here are the steps:

0. Label with '(S,0)' all vertices  $v_i$  that are not incident with an edge in  $M$ . Set variable  $step$  to 0.

Now repeat the next two steps until no vertex acquires a new label:

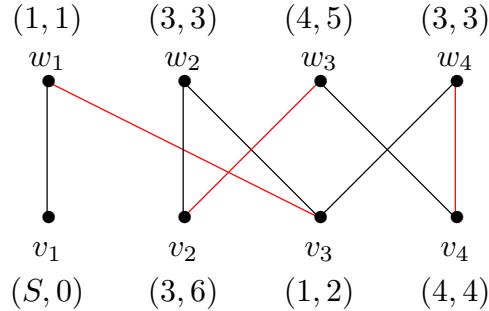
1. Increase  $step$  by 1. For each newly labeled vertex  $v_i$ , label with  $(i, step)$  any unlabeled neighbor  $w_j$  of  $v_i$  that is connected to  $v_i$  by an edge that is not in  $M$ .
2. Increase  $step$  by 1. For each newly labeled vertex  $w_i$ , label with  $(i, step)$  any unlabeled neighbor  $v_j$  of  $w_i$  that is connected to  $w_i$  by an edge in  $M$ .

Here “newly labeled” means labeled at the previous step. When labeling vertices in step 1 or 2, no vertex is given more than one label. For example, in step 1, it may be that  $w_k$  is a neighbor of the newly labeled vertices  $v_i$  and  $v_j$ . One of  $v_i$  and  $v_j$ , say  $v_i$ , will be considered first, and will cause  $w_k$  to be labeled; when  $v_j$  is considered,  $w_k$  is no longer unlabeled.

At the conclusion of the algorithm, if there is a labeled vertex  $w_i$  that is not incident with any edge of  $M$ , then there is an alternating chain, and we say the algorithm succeeds. If there is no such  $w_i$ , then there is no alternating chain, and we say the algorithm fails. The first of these claims is easy to see: Suppose vertex  $w_i$  is labeled  $(k_1, s)$ . It became labeled due to vertex  $v_{k_1}$  labeled  $(k_2, s - 1)$  that is connected by an edge not in  $M$  to  $w_i$ . In turn,  $v_{k_1}$  is connected by an edge in  $M$  to vertex  $w_{k_2}$  labeled  $(k_3, s - 2)$ . Continuing in this way, we discover an alternating chain ending at some vertex  $v_j$  labeled  $(S, 0)$ : since the second coordinate  $step$  decreases by 1 at each vertex along the chain, we cannot repeat vertices, and must eventually get to a vertex with  $step = 0$ . If we apply the algorithm to the graph in figure 4.5.2, we get the labeling shown in figure 4.5.5, which then identifies the alternating chain  $w_2, v_3, w_1, v_1$ . Note that as soon as a vertex  $w_i$  that is incident with no edge of  $M$  is labeled, we may stop, as there must be an alternating chain starting at  $w_i$ ; we need not continue the algorithm until no more labeling is possible. In the example in figure 4.5.5, we could stop after step 3, when  $w_2$  becomes labeled. Also, the  $step$  component of the labels is not really needed; it was included to make it easier to understand that if the algorithm succeeds, there really is an alternating chain.

To see that when the algorithm fails there is no alternating chain, we introduce a new concept.

**DEFINITION 4.5.3** A **vertex cover** in a graph is a set of vertices  $S$  such that every edge in the graph has at least one endpoint in  $S$ .  $\square$



**Figure 4.5.5** Labeling of a bipartite graph with matching;  $w_2, v_3, w_1, v_1$  is an alternating chain.

There is always a vertex cover of a graph, namely, the set of all the vertices of the graph. What is clearly more interesting is a smallest vertex cover, which is related to a maximum matching.

**THEOREM 4.5.4** If  $M$  is a matching in a graph and  $S$  is a vertex cover, then  $|M| \leq |S|$ .

**Proof.** Suppose  $\hat{M}$  is a matching of maximum size and  $\hat{S}$  is a vertex cover of minimum size. Since each edge of  $\hat{M}$  has an endpoint in  $\hat{S}$ , if  $|\hat{M}| > |\hat{S}|$  then some vertex in  $\hat{S}$  is incident with two edges of  $\hat{M}$ , a contradiction. Hence  $|M| \leq |\hat{M}| \leq |\hat{S}| \leq |S|$ . ■

Suppose that we have a matching  $M$  and vertex cover  $S$  for a graph, and that  $|M| = |S|$ . Then the theorem implies that  $M$  is a maximum matching and  $S$  is a minimum vertex cover. To show that when the algorithm fails there is no alternating chain, it is sufficient to show that there is a vertex cover that is the same size as  $M$ . Note that the proof of this theorem relies on the “official” version of the algorithm, that is, the algorithm continues until no new vertices are labeled.

**THEOREM 4.5.5** Suppose the algorithm fails on the bipartite graph  $G$  with matching  $M$ . Let  $U$  be the set of labeled  $w_i$ ,  $L$  the set of unlabeled  $v_i$ , and  $S = L \cup U$ . Then  $S$  is a vertex cover and  $|M| = |S|$ .

**Proof.** If  $S$  is not a cover, there is an edge  $\{v_i, w_j\}$  with neither  $v_i$  nor  $w_j$  in  $S$ , so  $v_i$  is labeled and  $w_j$  is not. If the edge is not in  $M$ , then the algorithm would have labeled  $w_j$  at the step after  $v_i$  became labeled, so the edge must be in  $M$ . Now  $v_i$  cannot be labeled  $(S, 0)$ , so  $v_i$  became labeled because it is connected to some labeled  $w_k$  by an edge of  $M$ . But now the two edges  $\{v_i, w_j\}$  and  $\{v_i, w_k\}$  are in  $M$ , a contradiction. So  $S$  is a vertex cover.

We know that  $|M| \leq |S|$ , so it suffices to show  $|S| \leq |M|$ , which we can do by finding an injection from  $S$  to  $M$ . Suppose that  $w_i \in S$ , so  $w_i$  is labeled. Since the algorithm failed,  $w_i$  is incident with an edge  $e$  of  $M$ ; let  $f(w_i) = e$ . If  $v_i \in S$ ,  $v_i$  is unlabeled; if  $v_i$

were not incident with any edge of  $M$ , then  $v_i$  would be labeled  $(S, 0)$ , so  $v_i$  is incident with an edge  $e$  of  $M$ ; let  $f(v_i) = e$ . Since  $G$  is bipartite, it is not possible that  $f(w_i) = f(w_j)$  or  $f(v_i) = f(v_j)$ . If  $f(w_i) = f(v_j)$ , then  $w_i$  and  $v_j$  are joined by an edge of  $M$ , and the algorithm would have labeled  $v_j$ . Hence,  $f$  is an injection. ■

We have now proved this theorem:

**THEOREM 4.5.6** In a bipartite graph  $G$ , the size of a maximum matching is the same as the size of a minimum vertex cover. ■

It is clear that the size of a maximum SDR is the same as the size of a maximum matching in the associated bipartite graph  $G$ . It is not too difficult to see directly that the size of a minimum vertex cover in  $G$  is the minimum value of  $f(n, i_1, i_2, \dots, i_k) = n - k + |\bigcup_{j=1}^k A_{i_j}|$ . Thus, if the size of a maximum matching is equal to the size of a minimum cover, then the size of a maximum SDR is equal to the minimum value of  $n - k + |\bigcup_{j=1}^k A_{i_j}|$ , and conversely. More concisely, theorem 4.5.6 is true if and only if theorem 4.2.1 is true.

More generally, in the schematic of figure 4.5.6, if any three of the relationships are known to be true, so is the fourth. In fact, we have proved all but the bottom equality, so we know it is true as well.

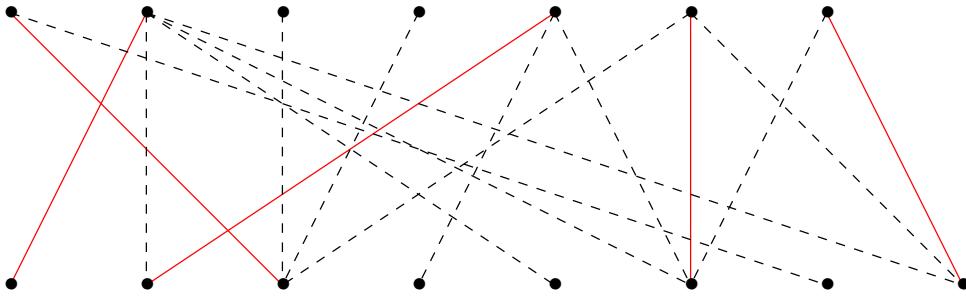
$$\begin{array}{ccc} \text{max sdr} & =? & \text{max matching} \\ & \parallel ? & \parallel ? \\ \text{min } f(n, \dots) & =? & \text{min cover} \end{array}$$

**Figure 4.5.6** If any three of the “=?” are “=”, so is the fourth.

Finally, note that we now have both a more efficient way to compute the size of a maximum SDR and a way to find the actual representatives: convert the SDR problem to the graph problem, find a maximum matching, and interpret the matching as an SDR.

### Exercises 4.5.

1. In this bipartite graph, find a maximum matching and a minimum vertex cover using the algorithm of this section. Start with the matching shown in red. Copies of this graph are available in [this pdf file](#).



2. Show directly that the size of a minimum vertex cover in  $G$  is the minimum value of  $n - k + |\bigcup_{j=1}^k A_{i_j}|$ , as mentioned above.

# 5

## Graph Theory

### 5.1 THE BASICS

See section 4.4 to review some basic terminology about graphs.

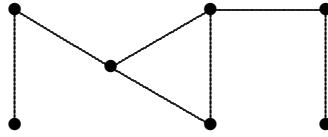
A graph  $G$  consists of a pair  $(V, E)$ , where  $V$  is the set of vertices and  $E$  the set of edges. We write  $V(G)$  for the vertices of  $G$  and  $E(G)$  for the edges of  $G$  when necessary to avoid ambiguity, as when more than one graph is under discussion.

If no two edges have the same endpoints we say there are no **multiple edges**, and if no edge has a single vertex as both endpoints we say there are no **loops**. A graph with no loops and no multiple edges is a **simple graph**. A graph with no loops, but possibly with multiple edges is a **multigraph**. The **condensation** of a multigraph is the simple graph formed by eliminating multiple edges, that is, removing all but one of the edges with the same endpoints. To form the condensation of a graph, all loops are also removed. We sometimes refer to a graph as a **general graph** to emphasize that the graph may have loops or multiple edges.

The edges of a simple graph can be represented as a set of two element sets; for example,

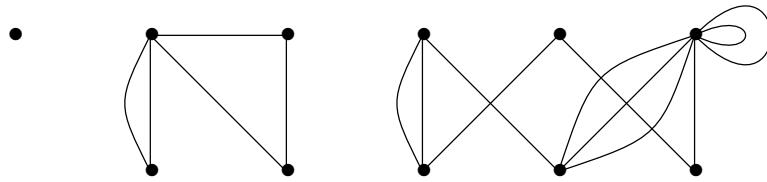
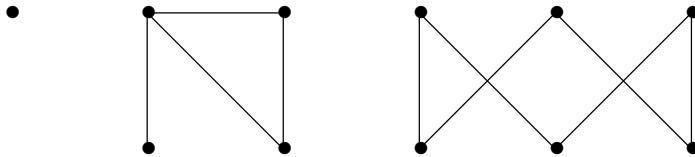
$$(\{v_1, \dots, v_7\}, \{\{v_1, v_2\}, \{v_2, v_3\}, \{v_3, v_4\}, \{v_3, v_5\}, \{v_4, v_5\}, \{v_5, v_6\}, \{v_6, v_7\}\})$$

is a graph that can be pictured as in figure 5.1.1. This graph is also a **connected** graph: each pair of vertices  $v, w$  is connected by a sequence of vertices and edges,  $v = v_1, e_1, v_2, e_2, \dots, v_k = w$ , where  $v_i$  and  $v_{i+1}$  are the endpoints of edge  $e_i$ . The graphs shown in figure 4.4.2 are connected, but the figure could be interpreted as a single graph that is not connected.

**Figure 5.1.1** A simple graph.

A graph  $G = (V, E)$  that is not simple can be represented by using multisets: a loop is a multiset  $\{v, v\} = \{2 \cdot v\}$  and multiple edges are represented by making  $E$  a multiset. The condensation of a multigraph may be formed by interpreting the multiset  $E$  as a set.

A general graph that is not connected, has loops, and has multiple edges is shown in figure 5.1.2. The condensation of this graph is shown in figure 5.1.3.

**Figure 5.1.2** A general graph: it is not connected and has loops and multiple edges.**Figure 5.1.3** The condensation of the previous graph.

The degree of a vertex  $v$ ,  $d(v)$ , is the number of times it appears as an endpoint of an edge. If there are no loops, this is the same as the number of edges incident with  $v$ , but if  $v$  is both endpoints of an edge, namely, of a loop, then this contributes 2 to the degree of  $v$ . The **degree sequence** of a graph is a list of its degrees; the order does not matter, but usually we list the degrees in increasing or decreasing order. The degree sequence of the graph in figure 5.1.2, listed clockwise starting at the upper left, is 0, 4, 2, 3, 2, 8, 2, 4, 3, 2, 2. We typically denote the degrees of the vertices of a graph by  $d_i$ ,  $i = 1, 2, \dots, n$ , where  $n$  is the number of vertices. Depending on context, the subscript  $i$  may match the subscript on a vertex, so that  $d_i$  is the degree of  $v_i$ , or the subscript may indicate the position of  $d_i$  in an increasing or decreasing list of the degrees; for example, we may state that the degree sequence is  $d_1 \leq d_2 \leq \dots \leq d_n$ .

Our first result, simple but useful, concerns the degree sequence.

**THEOREM 5.1.1** In any graph, the sum of the degree sequence is equal to twice the number of edges, that is,

$$\sum_{i=1}^n d_i = 2|E|.$$

**Proof.** Let  $d_i$  be the degree of  $v_i$ . The degree  $d_i$  counts the number of times  $v_i$  appears as an endpoint of an edge. Since each edge has two endpoints, the sum  $\sum_{i=1}^n d_i$  counts each edge twice. ■

An easy consequence of this theorem:

**COROLLARY 5.1.2** The number of odd numbers in a degree sequence is even. ■

An interesting question immediately arises: given a finite sequence of integers, is it the degree sequence of a graph? Clearly, if the sum of the sequence is odd, the answer is no. If the sum is even, it is not too hard to see that the answer is yes, provided we allow loops and multiple edges. The sequence need not be the degree sequence of a simple graph; for example, it is not hard to see that no simple graph has degree sequence 0, 1, 2, 3, 4. A sequence that is the degree sequence of a simple graph is said to be **graphical**. Graphical sequences have been characterized; the most well known characterization is given by this result:

**THEOREM 5.1.3** A sequence  $d_1 \geq d_2 \geq \dots \geq d_n$  is graphical if and only if  $\sum_{i=1}^n d_i$  is even and for all  $k \in \{1, 2, \dots, n\}$ ,

$$\sum_{i=1}^k d_i \leq k(k-1) + \sum_{i=k+1}^n \min(d_i, k).$$

■

It is not hard to see that if a sequence is graphical it has the property in the theorem; it is rather more difficult to see that any sequence with the property is graphical.

What does it mean for two graphs to be the same? Consider these three graphs:

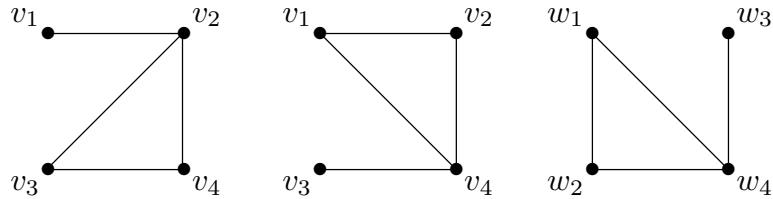
$$G_1 = (\{v_1, v_2, v_3, v_4\}, \{\{v_1, v_2\}, \{v_2, v_3\}, \{v_3, v_4\}, \{v_2, v_4\}\})$$

$$G_2 = (\{v_1, v_2, v_3, v_4\}, \{\{v_1, v_2\}, \{v_1, v_4\}, \{v_3, v_4\}, \{v_2, v_4\}\})$$

$$G_3 = (\{w_1, w_2, w_3, w_4\}, \{\{w_1, w_2\}, \{w_1, w_4\}, \{w_3, w_4\}, \{w_2, w_4\}\})$$

These are pictured in figure 5.1.4. Simply looking at the lists of vertices and edges, they don't appear to be the same. Looking more closely,  $G_2$  and  $G_3$  are the same except for

the names used for the vertices:  $v_i$  in one case,  $w_i$  in the other. Looking at the pictures, there is an obvious sense in which all three are the same: each is a triangle with an edge (and vertex) dangling from one of the three vertices. Although  $G_1$  and  $G_2$  use the same names for the vertices, they apply to different vertices in the graph: in  $G_1$  the “dangling” vertex (officially called a **pendant** vertex) is called  $v_1$ , while in  $G_2$  it is called  $v_3$ . Finally, note that in the figure,  $G_2$  and  $G_3$  look different, even though they are clearly the same based on the vertex and edge lists.



**Figure 5.1.4** Three isomorphic graphs.

So how should we define “sameness” for graphs? We use a familiar term and definition: isomorphism.

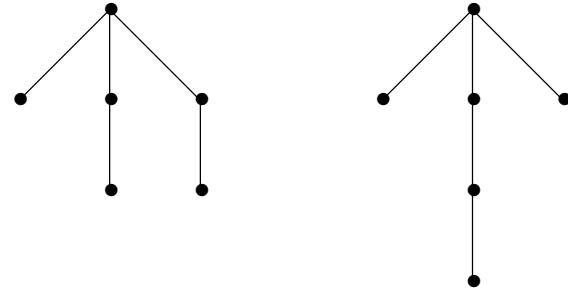
**DEFINITION 5.1.4** Suppose  $G_1 = (V, E)$  and  $G_2 = (W, F)$ .  $G_1$  and  $G_2$  are **isomorphic** if there is a bijection  $f: V \rightarrow W$  such that  $\{v_1, v_2\} \in E$  if and only if  $\{f(v_1), f(v_2)\} \in F$ . In addition, the repetition numbers of  $\{v_1, v_2\}$  and  $\{f(v_1), f(v_2)\}$  are the same if multiple edges or loops are allowed. This bijection  $f$  is called an **isomorphism**. When  $G_1$  and  $G_2$  are isomorphic, we write  $G_1 \cong G_2$ .  $\square$

Each pair of graphs in figure 5.1.4 are isomorphic. For example, to show explicitly that  $G_1 \cong G_3$ , an isomorphism is

$$\begin{aligned} f(v_1) &= w_3 \\ f(v_2) &= w_4 \\ f(v_3) &= w_2 \\ f(v_4) &= w_1. \end{aligned}$$

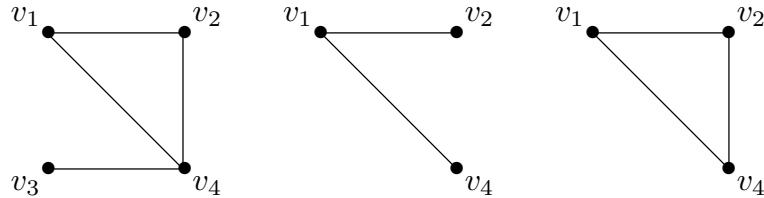
Clearly, if two graphs are isomorphic, their degree sequences are the same. The converse is not true; the graphs in figure 5.1.5 both have degree sequence 1, 1, 1, 2, 2, 3, but in one the degree-2 vertices are adjacent to each other, while in the other they are not. In general, if two graphs are isomorphic, they share all “graph theoretic” properties, that is, properties that depend only on the graph. As an example of a non-graph theoretic property, consider “the number of times edges cross when the graph is drawn in the plane.”

In a more or less obvious way, some graphs are contained in others.



**Figure 5.1.5** Non-isomorphic graphs with degree sequence 1, 1, 1, 2, 2, 3.

**DEFINITION 5.1.5** Graph  $H = (W, F)$  is a **subgraph** of graph  $G = (V, E)$  if  $W \subseteq V$  and  $F \subseteq E$ . (Since  $H$  is a graph, the edges in  $F$  have their endpoints in  $W$ .)  $H$  is an **induced subgraph** if  $F$  consists of all edges in  $E$  with endpoints in  $W$ . See figure 5.1.6. Whenever  $U \subseteq V$  we denote the induced subgraph of  $G$  on vertices  $U$  as  $G[U]$ .  $\square$



**Figure 5.1.6** Left to right: a graph, a subgraph, an induced subgraph.

A path in a graph is a subgraph that is a path; if the endpoints of the path are  $v$  and  $w$  we say it is a path from  $v$  to  $w$ . A cycle in a graph is a subgraph that is a cycle. A **clique** in a graph is a subgraph that is a complete graph.

If a graph  $G$  is not connected, define  $v \sim w$  if and only if there is a path connecting  $v$  and  $w$ . It is not hard to see that this is an equivalence relation. Each equivalence class corresponds to an induced subgraph  $G$ ; these subgraphs are called the **connected components** of the graph.

### Exercises 5.1.

1. The complement  $\overline{G}$  of the simple graph  $G$  is a simple graph with the same vertices as  $G$ , and  $\{v, w\}$  is an edge of  $\overline{G}$  if and only if it is not an edge of  $G$ . A graph  $G$  is self-complementary if  $G \cong \overline{G}$ . Show that if  $G$  is self-complementary then it has  $4k$  or  $4k + 1$  vertices for some  $k$ . Find self-complementary graphs on 4 and 5 vertices.
2. Prove that if  $\sum_{i=1}^n d_i$  is even, there is a graph (not necessarily simple) with degree sequence  $d_1, d_2, \dots, d_n$ .
3. Suppose  $d_1 \geq d_2 \geq \dots \geq d_n$  and  $\sum_{i=1}^n d_i$  is even. Prove that there is a multigraph (no loops) with degree sequence  $d_1, d_2, \dots, d_n$  if and only if  $d_1 \leq \sum_{i=2}^n d_i$ .
4. Prove that 0, 1, 2, 3, 4 is not graphical.

## 96 Chapter 5 Graph Theory

5. Is  $4, 4, 3, 2, 2, 1, 1$  graphical? If not, explain why; if so, find a simple graph with this degree sequence.
6. Is  $4, 4, 4, 2, 2$  graphical? If not, explain why, and find a multigraph (no loops) with this degree sequence; if so, find a simple graph with this degree sequence.
7. Prove that a simple graph with  $n \geq 2$  vertices has two vertices of the same degree.
8. Prove the “only if” part of theorem 5.1.3.
9. Show that the condition on the degrees in theorem 5.1.3 is equivalent to this condition:  $\sum_{i=1}^n d_i$  is even and for all  $k \in \{1, 2, \dots, n\}$ , and all  $\{i_1, i_2, \dots, i_k\} \subseteq [n]$ ,

$$\sum_{j=1}^k d_{i_j} \leq k(k-1) + \sum_{i \notin \{i_1, i_2, \dots, i_k\}} \min(d_i, k).$$

Do not use theorem 5.1.3.

10. Draw the 11 non-isomorphic graphs with four vertices.
11. Suppose  $G_1 \cong G_2$ . Show that if  $G_1$  contains a cycle of length  $k$  so does  $G_2$ .
12. Define  $v \sim w$  if and only if there is a path connecting vertices  $v$  and  $w$ . Prove that  $\sim$  is an equivalence relation.
13. Prove the “if” part of theorem 5.1.3, as follows:

The proof is by induction on  $s = \sum_{i=1}^n d_i$ . This is easy to see if  $s = 2$ , so suppose  $s > 2$ . Without loss of generality we may suppose that  $d_n > 0$ . Let  $t$  be the least integer such that  $d_t > d_{t+1}$ , or  $t = n - 1$  if there is no such integer. Let  $d'_t = d_t - 1$ ,  $d'_n = d_n - 1$ , and  $d'_i = d_i$  for all other  $i$ . Note that  $d'_1 \geq d'_2 \geq \dots \geq d'_n$ . We want to show that the sequence  $\{d'_i\}$  satisfies the condition of the theorem, that is, that for all  $k \in \{1, 2, \dots, n\}$ ,

$$\sum_{i=1}^k d'_i \leq k(k-1) + \sum_{i=k+1}^n \min(d'_i, k).$$

There are five cases:

1.  $k \geq t$
2.  $k < t$ ,  $d_k < k$
3.  $k < t$ ,  $d_k = k$
4.  $k < t$ ,  $d_n > k$
5.  $k < t$ ,  $d_k > k$ ,  $d_n \leq k$

By the induction hypothesis, there is a simple graph with degree sequence  $\{d'_i\}$ . Finally, show that there is a graph with degree sequence  $\{d_i\}$ .

This proof is due to S. A. Choudum, *A Simple Proof of the Erdős-Gallai Theorem on Graph Sequences*, Bulletin of the Australian Mathematics Society, vol. 33, 1986, pp. 67-70. The proof by Paul Erdős and Tibor Gallai was long; Berge provided a shorter proof that used results in the theory of network flows. Choudum’s proof is both short and elementary.

## 5.2 EULER CIRCUITS AND WALKS

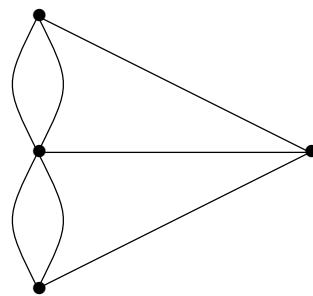
The first problem in graph theory dates to 1735, and is called the [Seven Bridges of Königsberg](#). In Königsberg were two islands, connected to each other and the mainland

by seven bridges, as shown in figure 5.2.1. The question, which made its way to Euler, was whether it was possible to take a walk and cross over each bridge exactly once; Euler showed that it is not possible.



**Figure 5.2.1** The Seven Bridges of Königsberg.

We can represent this problem as a graph, as in figure 5.2.2.



**Figure 5.2.2** The Seven Bridges of Königsberg as a graph.

The two sides of the river are represented by the top and bottom vertices, and the islands by the middle two vertices. There are two possible interpretations of the question, depending on whether the goal is to end the walk at its starting point. Perhaps inspired by this problem, a **walk** in a graph is defined as follows.

**DEFINITION 5.2.1** A walk in a graph is a sequence of vertices and edges,

$$v_1, e_1, v_2, e_2, \dots, v_k, e_k, v_{k+1}$$

such that the endpoints of edge  $e_i$  are  $v_i$  and  $v_{i+1}$ . In general, the edges and vertices may appear in the sequence more than once. If  $v_1 = v_{k+1}$ , the walk is a **closed walk** or a **circuit**.  $\square$

We will deal first with the case in which the walk is to start and end at the same place. A successful walk in Königsberg corresponds to a closed walk in the graph in which every edge is used exactly once.

What can we say about this walk in the graph, or indeed a closed walk in any graph that uses every edge exactly once? Such a walk is called an **Euler circuit**. If there are no vertices of degree 0, the graph must be connected, as this one is. Beyond that, imagine tracing out the vertices and edges of the walk on the graph. At every vertex other than the common starting and ending point, we come into the vertex along one edge and go out along another; this can happen more than once, but since we cannot use edges more than once, the number of edges incident at such a vertex must be even. Already we see that we're in trouble in this particular graph, but let's continue the analysis. The common starting and ending point may be visited more than once; except for the very first time we leave the starting vertex, and the last time we arrive at the vertex, each such visit uses exactly two edges. Together with the edges used first and last, this means that the starting vertex must also have even degree. Thus, since the Königsberg Bridges graph has odd degrees, the desired walk does not exist.

The question that should immediately spring to mind is this: if a graph is connected and the degree of every vertex is even, is there an Euler circuit? The answer is yes.

**THEOREM 5.2.2** If  $G$  is a connected graph, then  $G$  contains an Euler circuit if and only if every vertex has even degree.

**Proof.** We have already shown that if there is an Euler circuit, all degrees are even.

We prove the other direction by induction on the number of edges. If  $G$  has no edges the problem is trivial, so we assume that  $G$  has edges.

We start by finding some closed walk that does not use any edge more than once: Start at any vertex  $v_0$ ; follow any edge from this vertex, and continue to do this at each new vertex, that is, upon reaching a vertex, choose some unused edge leading to another vertex. Since every vertex has even degree, it is always possible to leave a vertex at which we arrive, until we return to the starting vertex, and every edge incident with the starting vertex has been used. The sequence of vertices and edges formed in this way is a closed walk; if it uses every edge, we are done.

Otherwise, form graph  $G'$  by removing all the edges of the walk.  $G'$  is not connected, since vertex  $v_0$  is not incident with any remaining edge. The rest of the graph, that is,  $G'$  without  $v_0$ , may or may not be connected. It consists of one or more connected subgraphs, each with fewer edges than  $G$ ; call these graphs  $G_1, G_2, \dots, G_k$ . Note that when we remove the edges of the initial walk, we reduce the degree of every vertex by an even number, so all the vertices of each graph  $G_i$  have even degree. By the induction hypothesis, each  $G_i$  has an Euler circuit. These closed walks together with the original closed walk use every edge of  $G$  exactly once.

Suppose the original closed walk is  $v_0, v_1, \dots, v_m = v_0$ , abbreviated to leave out the edges. Because  $G$  is connected, at least one vertex in each  $G_i$  appears in this sequence, say vertices  $w_{1,1} \in G_1, w_{2,1} \in G_2, \dots, w_{k,1} \in G_k$ , listed in the order they appear in  $v_0, v_1, \dots, v_m$ . The Euler circuits of the graphs  $G_i$  are

$$w_{1,1}, w_{1,2}, \dots, w_{1,m_1} = w_{1,1}$$

$$w_{2,1}, w_{2,2}, \dots, w_{2,m_2} = w_{2,1}$$

⋮

$$w_{k,1}, w_{k,2}, \dots, w_{k,m_k} = w_{k,1}.$$

By pasting together the original closed walk with these, we form a closed walk in  $G$  that uses every edge exactly once:

$$\begin{aligned} v_0, v_1, \dots, v_{i_1} &= w_{1,1}, w_{1,2}, \dots, w_{1,m_1} = v_{i_1}, v_{i_1+1}, \\ \dots, v_{i_2} &= w_{2,1}, \dots, w_{2,m_2} = v_{i_2}, v_{i_2+1}, \\ \dots, v_{i_k} &= w_{k,1}, \dots, w_{k,m_k} = v_{i_k}, v_{i_k+1}, \dots, v_m = v_0. \end{aligned}$$

■

Now let's turn to the second interpretation of the problem: is it possible to walk over all the bridges exactly once, if the starting and ending points need not be the same? In a graph  $G$ , a walk that uses all of the edges but is not an Euler circuit is called an **Euler walk**. It is not too difficult to do an analysis much like the one for Euler circuits, but it is even easier to use the Euler circuit result itself to characterize Euler walks.

**THEOREM 5.2.3** A connected graph  $G$  has an Euler walk if and only if exactly two vertices have odd degree.

**Proof.** Suppose first that  $G$  has an Euler walk starting at vertex  $v$  and ending at vertex  $w$ . Add a new edge to the graph with endpoints  $v$  and  $w$ , forming  $G'$ .  $G'$  has an Euler circuit, and so by the previous theorem every vertex has even degree. The degrees of  $v$  and  $w$  in  $G$  are therefore odd, while all others are even.

Now suppose that the degrees of  $v$  and  $w$  in  $G$  are odd, while all other vertices have even degree. Add a new edge  $e$  to the graph with endpoints  $v$  and  $w$ , forming  $G'$ . Every vertex in  $G'$  has even degree, so by the previous theorem there is an Euler circuit which we can write as

$$v, e_1, v_2, e_2, \dots, w, e, v,$$

so that

$$v, e_1, v_2, e_2, \dots, w$$

is an Euler walk. ■

### **Exercises 5.2.**

1. Suppose a connected graph  $G$  has degree sequence  $d_1, d_2, \dots, d_n$ . How many edges must be added to  $G$  so that the resulting graph has an Euler circuit? Explain.
2. Which complete graphs  $K_n$ ,  $n \geq 2$ , have Euler circuits? Which have Euler walks? Justify your answers.
3. Prove that if vertices  $v$  and  $w$  are joined by a walk they are joined by a path.
4. Show that if  $G$  is connected and has exactly  $2k$  vertices of odd degree,  $k \geq 1$ , its edges can be partitioned into  $k$  walks. Is this true for non-connected  $G$ ?

## **5.3 HAMILTON CYCLES AND PATHS**

Here is a problem similar to the Königsberg Bridges problem: suppose a number of cities are connected by a network of roads. Is it possible to visit all the cities exactly once, without traveling any road twice? We assume that these roads do not intersect except at the cities. Again there are two versions of this problem, depending on whether we want to end at the same city in which we started.

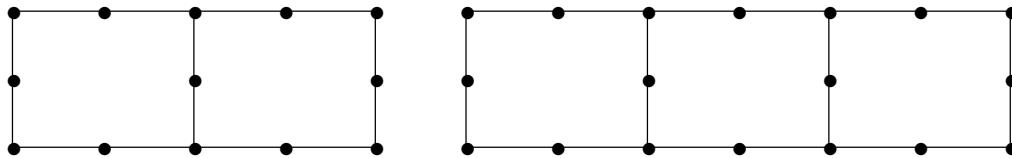
This problem can be represented by a graph: the vertices represent cities, the edges represent the roads. We want to know if this graph has a cycle, or path, that uses every vertex exactly once. (Recall that a cycle in a graph is a subgraph that is a cycle, and a path is a subgraph that is a path.) There is no benefit or drawback to loops and multiple edges in this context: loops can never be used in a Hamilton cycle or path (except in the trivial case of a graph with a single vertex), and at most one of the edges between two vertices can be used. So we assume for this discussion that all graphs are simple.

**DEFINITION 5.3.1** A cycle that uses every vertex in a graph exactly once is called a **Hamilton cycle**, and a path that uses every vertex in a graph exactly once is called a **Hamilton path**. □

Unfortunately, this problem is much more difficult than the corresponding Euler circuit and walk problems; there is no good characterization of graphs with Hamilton paths and cycles. Note that if a graph has a Hamilton cycle then it also has a Hamilton path.

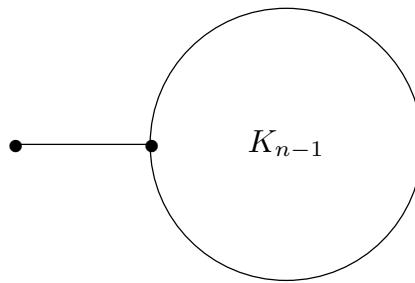
There are some useful conditions that imply the existence of a Hamilton cycle or path, which typically say in some form that there are many edges in the graph. An extreme example is the complete graph  $K_n$ : it has as many edges as any simple graph on  $n$  vertices can have, and it has many Hamilton cycles.

The problem for a characterization is that there are graphs with Hamilton cycles that do not have very many edges. The simplest is a cycle,  $C_n$ : this has only  $n$  edges but has a Hamilton cycle. On the other hand, figure 5.3.1 shows graphs with just a few more edges than the cycle on the same number of vertices, but without Hamilton cycles.



**Figure 5.3.1** A graph with a Hamilton path but not a Hamilton cycle, and one with neither.

There are also graphs that seem to have many edges, yet have no Hamilton cycle, as indicated in figure 5.3.2.



**Figure 5.3.2** A graph with many edges but no Hamilton cycle: a complete graph  $K_{n-1}$  joined by an edge to a single vertex. This graph has  $\binom{n-1}{2} + 1$  edges.

The key to a successful condition sufficient to guarantee the existence of a Hamilton cycle is to require many edges at lots of vertices.

**THEOREM 5.3.2 (Ore)** If  $G$  is a simple graph on  $n$  vertices,  $n \geq 3$ , and  $d(v) + d(w) \geq n$  whenever  $v$  and  $w$  are not adjacent, then  $G$  has a Hamilton cycle.

**Proof.** First we show that  $G$  is connected. If not, let  $v$  and  $w$  be vertices in two different connected components of  $G$ , and suppose the components have  $n_1$  and  $n_2$  vertices. Then

$d(v) \leq n_1 - 1$  and  $d(w) \leq n_2 - 1$ , so  $d(v) + d(w) \leq n_1 + n_2 - 2 < n$ . But since  $v$  and  $w$  are not adjacent, this is a contradiction.

Now consider a longest possible path in  $G$ :  $v_1, v_2, \dots, v_k$ . Suppose, for a contradiction, that  $k < n$ , so there is some vertex  $w$  adjacent to one of  $v_2, v_3, \dots, v_{k-1}$ , say to  $v_i$ . If  $v_1$  is adjacent to  $v_k$ , then  $w, v_i, v_{i+1}, \dots, v_k, v_1, v_2, \dots, v_{i-1}$  is a path of length  $k+1$ , a contradiction. Hence,  $v_1$  is not adjacent to  $v_k$ , and so  $d(v_1) + d(v_k) \geq n$ . The neighbors of  $v_1$  are among  $\{v_2, v_3, \dots, v_{k-1}\}$  as are the neighbors of  $v_k$ . Consider the vertices

$$W = \{v_{l+1} \mid v_l \text{ is a neighbor of } v_k\}.$$

Then  $|N(v_k)| = |W|$  and  $W \subseteq \{v_3, v_4, \dots, v_k\}$  and  $N(v_1) \subseteq \{v_2, v_3, \dots, v_{k-1}\}$ , so  $W \cup N(v_1) \subseteq \{v_2, v_3, \dots, v_k\}$ , a set with  $k-1 < n$  elements. Since  $|N(v_1)| + |W| = |N(v_1)| + |N(v_k)| \geq n$ ,  $N(v_1)$  and  $W$  must have a common element,  $v_j$ ; note that  $3 \leq j \leq k-1$ . Then this is a cycle of length  $k$ :

$$v_1, v_j, v_{j+1}, \dots, v_k, v_{j-1}, v_{j-2}, \dots, v_1.$$

We can relabel the vertices for convenience:

$$v_1 = w_1, w_2, \dots, w_k = v_2, w_1.$$

Now as before,  $w$  is adjacent to some  $w_l$ , and  $w, w_l, w_{l+1}, \dots, w_k, w_1, w_2, \dots, w_{l-1}$  is a path of length  $k+1$ , a contradiction. Thus,  $k = n$ , and, renumbering the vertices for convenience, we have a Hamilton path  $v_1, v_2, \dots, v_n$ . If  $v_1$  is adjacent to  $v_n$ , there is a Hamilton cycle, as desired.

If  $v_1$  is not adjacent to  $v_n$ , the neighbors of  $v_1$  are among  $\{v_2, v_3, \dots, v_{n-1}\}$  as are the neighbors of  $v_n$ . Consider the vertices

$$W = \{v_{l+1} \mid v_l \text{ is a neighbor of } v_n\}.$$

Then  $|N(v_n)| = |W|$  and  $W \subseteq \{v_3, v_4, \dots, v_n\}$ , and  $N(v_1) \subseteq \{v_2, v_3, \dots, v_{n-1}\}$ , so  $W \cup N(v_1) \subseteq \{v_2, v_3, \dots, v_n\}$ , a set with  $n-1 < n$  elements. Since  $|N(v_1)| + |W| = |N(v_1)| + |N(v_n)| \geq n$ ,  $N(v_1)$  and  $W$  must have a common element,  $v_i$ ; note that  $3 \leq i \leq n-1$ . Then this is a cycle of length  $n$ :

$$v_1, v_i, v_{i+1}, \dots, v_k, v_{i-1}, v_{i-2}, \dots, v_1,$$

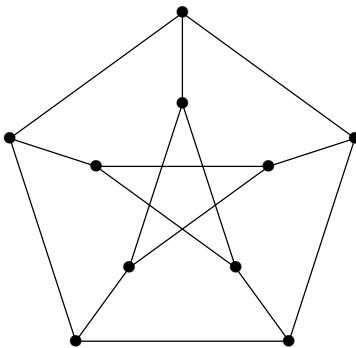
and is a Hamilton cycle. ■

The property used in this theorem is called the **Ore property**; if a graph has the Ore property it also has a Hamilton path, but we can weaken the condition slightly if our goal is to show there is a Hamilton path. The proof of this theorem is nearly identical to the preceding proof.

**THEOREM 5.3.3** If  $G$  is a simple graph on  $n$  vertices and  $d(v) + d(w) \geq n-1$  whenever  $v$  and  $w$  are not adjacent, then  $G$  has a Hamilton path. ■

**Exercises 5.3.**

1. Suppose a simple graph  $G$  on  $n$  vertices has at least  $\frac{(n-1)(n-2)}{2} + 2$  edges. Prove that  $G$  has a Hamilton cycle. For  $n \geq 2$ , show that there is a simple graph with  $\frac{(n-1)(n-2)}{2} + 1$  edges that has no Hamilton cycle.
2. Prove theorem 5.3.3.
3. The graph shown below is the **Petersen** graph. Does it have a Hamilton cycle? Justify your answer. Does it have a Hamilton path? Justify your answer.



## 5.4 BIPARTITE GRAPHS

We have already seen how bipartite graphs arise naturally in some circumstances. Here we explore bipartite graphs a bit more.

It is easy to see that all closed walks in a bipartite graph must have even length, since the vertices along the walk must alternate between the two parts. Remarkably, the converse is true. We need one new definition:

**DEFINITION 5.4.1** The distance between vertices  $v$  and  $w$ ,  $d(v, w)$ , is the length of a shortest walk between the two. If there is no walk between  $v$  and  $w$ , the distance is undefined.  $\square$

**THEOREM 5.4.2**  $G$  is bipartite if and only if all closed walks in  $G$  are of even length.

*Proof.* The forward direction is easy, as discussed above.

Now suppose that all closed walks have even length. We may assume that  $G$  is connected; if not, we deal with each connected component separately.

Let  $v$  be a vertex of  $G$ , let  $X$  be the set of all vertices at even distance from  $v$ , and  $Y$  be the set of vertices at odd distance from  $v$ . We claim that all edges of  $G$  join a vertex of  $X$  to a vertex of  $Y$ . Suppose not; then there are adjacent vertices  $u$  and  $w$  such that  $d(v, u)$  and  $d(v, w)$  have the same parity. Then there is a closed walk from  $v$  to  $u$  to  $w$  to  $v$  of length  $d(v, u) + 1 + d(v, w)$ , which is odd, a contradiction. ■

The closed walk that provides the contradiction is not necessarily a cycle, but this can be remedied, providing a slightly different version of the theorem.

**COROLLARY 5.4.3**  $G$  is bipartite if and only if all cycles in  $G$  are of even length.

**Proof.** Again the forward direction is easy, and again we assume  $G$  is connected. As before, let  $v$  be a vertex of  $G$ , let  $X$  be the set of all vertices at even distance from  $v$ , and  $Y$  be the set of vertices at odd distance from  $v$ . If two vertices in  $X$  are adjacent, or two vertices in  $Y$  are adjacent, then as in the previous proof, there is a closed walk of odd length.

To finish the proof, it suffices to show that if there is a closed walk  $W$  of odd length then there is a cycle of odd length. The proof is by induction on the length of the closed walk.

If  $W$  has no repeated vertices, we are done. Otherwise, suppose the closed walk is

$$v = v_1, e_1, \dots, v_i = v, \dots, v_k = v = v_1.$$

Then

$$v = v_1, \dots, v_i = v \quad \text{and} \quad v = v_i, e_i, v_{i+1}, \dots, v_k = v$$

are closed walks, both are shorter than the original closed walk, and one of them has odd length. By the induction hypothesis, there is a cycle of odd length. ■

It is frequently fruitful to consider graph properties in the limited context of bipartite graphs (or other special types of graph). For example, what can we say about Hamilton cycles in simple bipartite graphs? Suppose the partition of the vertices of the bipartite graph is  $X$  and  $Y$ . Because any cycle alternates between vertices of the two parts of the bipartite graph, if there is a Hamilton cycle then  $|X| = |Y| \geq 2$ . In such a case, the degree of every vertex is at most  $n/2$ , where  $n$  is the number of vertices, namely  $n = |X| + |Y|$ . Thus the Ore condition ( $d(v) + d(w) \geq n$  when  $v$  and  $w$  are not adjacent) is equivalent to  $d(v) = n/2$  for all  $v$ . This means the only simple bipartite graph that satisfies the Ore condition is the **complete bipartite graph**  $K_{n/2, n/2}$ , in which the two parts have size  $n/2$  and every vertex of  $X$  is adjacent to every vertex of  $Y$ . The upshot is that the Ore property gives no interesting information about bipartite graphs.

Of course, as with more general graphs, there are bipartite graphs with few edges and a Hamilton cycle: any even length cycle is an example.

We note that, in general, a complete bipartite graph  $K_{m,n}$  is a bipartite graph with  $|X| = m$ ,  $|Y| = n$ , and every vertex of  $X$  is adjacent to every vertex of  $Y$ . The only such graphs with Hamilton cycles are those in which  $m = n$ .

### Exercises 5.4.

1. Prove that there is a bipartite multigraph with degree sequence  $d_1, \dots, d_n$  if and only if there is a partition  $\{I, J\}$  of  $[n]$  such that

$$\sum_{i \in I} d_i = \sum_{i \in J} d_i.$$

2. What is the smallest number of edges that can be removed from  $K_5$  to create a bipartite graph?
3. A **regular graph** is one in which the degree of every vertex is the same. Show that if  $G$  is a regular bipartite graph, and the common degree of the vertices is at least 1, then the two parts are the same size.
4. A **perfect matching** is one in which all vertices of the graph are incident with exactly one edge in the matching. Show that a regular bipartite graph with common degree at least 1 has a perfect matching. (We discussed matchings in section 4.5.)

## 5.5 TREES

Another useful special class of graphs:

**DEFINITION 5.5.1** A connected graph  $G$  is a **tree** if it is **acyclic**, that is, it has no cycles. More generally, an acyclic graph is called a **forest**.  $\square$

Two small examples of trees are shown in figure 5.1.5. Note that the definition implies that no tree has a loop or multiple edges.

**THEOREM 5.5.2** Every tree  $T$  is bipartite.

**Proof.** Since  $T$  has no cycles, it is true that every cycle of  $T$  has even length. By corollary 5.4.3,  $T$  is bipartite.  $\blacksquare$

**DEFINITION 5.5.3** A vertex of degree one is called a **pendant vertex**, and the edge incident to it is a **pendant edge**.  $\square$

**THEOREM 5.5.4** Every tree on two or more vertices has at least one pendant vertex.

**Proof.** We prove the contrapositive. Suppose graph  $G$  has no pendant vertices. Starting at any vertex  $v$ , follow a sequence of distinct edges until a vertex repeats; this is possible because the degree of every vertex is at least two, so upon arriving at a vertex for the first time it is always possible to leave the vertex on another edge. When a vertex repeats for the first time, we have discovered a cycle.  $\blacksquare$

This theorem often provides the key step in an induction proof, since removing a pendant vertex (and its pendant edge) leaves a smaller tree.

**THEOREM 5.5.5** A tree on  $n$  vertices has exactly  $n - 1$  edges.

**Proof.** A tree on 1 vertex has 0 edges; this is the base case.

If  $T$  is a tree on  $n \geq 2$  vertices, it has a pendant vertex. Remove this vertex and its pendant edge to get a tree  $T'$  on  $n - 1$  vertices. By the induction hypothesis,  $T'$  has  $n - 2$  edges; thus  $T$  has  $n - 1$  edges. ■

**THEOREM 5.5.6** A tree with a vertex of degree  $k \geq 1$  has at least  $k$  pendant vertices. In particular, every tree on at least two vertices has at least two pendant vertices.

**Proof.** The case  $k = 1$  is obvious. Let  $T$  be a tree with  $n$  vertices, degree sequence  $\{d_i\}_{i=1}^n$ , and a vertex of degree  $k \geq 2$ , and let  $l$  be the number of pendant vertices. Without loss of generality,  $1 = d_1 = d_2 = \dots = d_l$  and  $d_{l+1} = k$ . Then

$$2(n - 1) = \sum_{i=1}^n d_i = l + k + \sum_{i=l+2}^n d_i \geq l + k + 2(n - l - 1).$$

This reduces to  $l \geq k$ , as desired.

If  $T$  is a tree on two vertices, each of the vertices has degree 1. If  $T$  has at least three vertices it must have a vertex of degree  $k \geq 2$ , since otherwise  $2(n - 1) = \sum_{i=1}^n d_i = n$ , which implies  $n = 2$ . Hence it has at least  $k \geq 2$  pendant vertices. ■

Trees are quite useful in their own right, but also for the study of general graphs.

**DEFINITION 5.5.7** If  $G$  is a connected graph on  $n$  vertices, a **spanning tree** for  $G$  is a subgraph of  $G$  that is a tree on  $n$  vertices. □

**THEOREM 5.5.8** Every connected graph has a spanning tree.

**Proof.** By induction on the number of edges. If  $G$  is connected and has zero edges, it is a single vertex, so  $G$  is already a tree.

Now suppose  $G$  has  $m \geq 1$  edges. If  $G$  is a tree, it is its own spanning tree. Otherwise,  $G$  contains a cycle; remove one edge of this cycle. The resulting graph  $G'$  is still connected and has fewer edges, so it has a spanning tree; this is also a spanning tree for  $G$ . ■

In general, spanning trees are not unique, that is, a graph may have many spanning trees. It is possible for some edges to be in every spanning tree even if there are multiple spanning trees. For example, any pendant edge must be in every spanning tree, as must any edge whose removal disconnects the graph (such an edge is called a **bridge**.)

**COROLLARY 5.5.9** If  $G$  is connected, it has at least  $n - 1$  edges; moreover, it has exactly  $n - 1$  edges if and only if it is a tree.

**Proof.** If  $G$  is connected, it has a spanning tree, which has  $n - 1$  edges, all of which are edges of  $G$ .

If  $G$  has  $n - 1$  edges, which must be the edges of its spanning tree, then  $G$  is a tree. ■

**THEOREM 5.5.10**  $G$  is a tree if and only if there is a unique path between any two vertices.

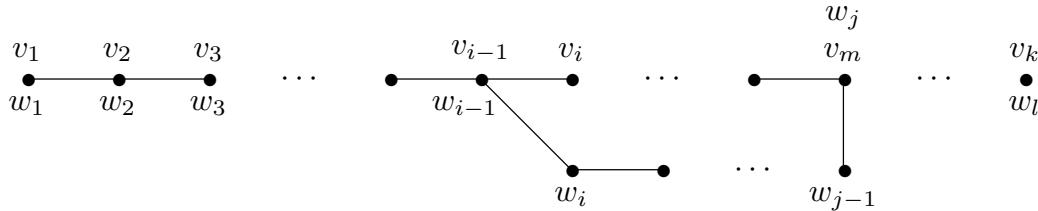
**Proof.**

**if:** Since every two vertices are connected by a path,  $G$  is connected. For a contradiction, suppose there is a cycle in  $G$ ; then any two vertices on the cycle are connected by at least two distinct paths, a contradiction.

**only if:** If  $G$  is a tree it is connected, so between any two vertices there is at least one path. For a contradiction, suppose there are two different paths from  $v$  to  $w$ :

$$v = v_1, v_2, \dots, v_k = w \quad \text{and} \quad v = w_1, w_2, \dots, w_l = w.$$

Let  $i$  be the smallest integer such that  $v_i \neq w_i$ . Then let  $j$  be the smallest integer greater than or equal to  $i$  such that  $w_j = v_m$  for some  $m$ , which must be at least  $i$ . (Since  $w_l = v_k$ , such an  $m$  must exist.) Then  $v_{i-1}, v_i, \dots, v_m = w_j, w_{j-1}, \dots, w_{i-1} = v_{i-1}$  is a cycle in  $G$ , a contradiction. See figure 5.5.1. ■



**Figure 5.5.1** Distinct paths imply the existence of a cycle.

**DEFINITION 5.5.11** A **cutpoint** in a connected graph  $G$  is a vertex whose removal disconnects the graph. □

**THEOREM 5.5.12** Every connected graph has a vertex that is not a cutpoint.

**Proof.** Remove a pendant vertex in a spanning tree for the graph. ■

### Exercises 5.5.

1. Suppose that  $G$  is a connected graph, and that every spanning tree contains edge  $e$ . Show that  $e$  is a bridge.
2. Show that every edge in a tree is a bridge.

3. Show that  $G$  is a tree if and only if it has no cycles and adding any new edge creates a graph with exactly one cycle.
4. Which trees have Euler walks?
5. Which trees have Hamilton paths?
6. Let  $n \geq 2$ . Show that there is a tree with degree sequence  $d_1, d_2, \dots, d_n$  if and only if  $d_i > 0$  for all  $i$  and  $\sum_{i=1}^n d_i = 2(n - 1)$ .
7. A multitree is a multigraph whose condensation is a tree. Let  $n \geq 2$ . Let  $d_1, d_2, \dots, d_n$  be positive integers, and let  $g$  be the greatest common divisor of the  $d_i$ . Show that there is a multitree with degree sequence  $d_1, d_2, \dots, d_n$  if and only if  $\sum_{i=1}^n d_i/g \geq 2(n - 1)$  and for some partition  $I, J$  of  $[n]$ ,  $\sum_{i \in I} d_i = \sum_{i \in J} d_i$ .

## 5.6 OPTIMAL SPANNING TREES

In some applications, a graph  $G$  is augmented by associating a weight or cost with each edge; such a graph is called a **weighted graph**. For example, if a graph represents a network of roads, the weight of an edge might be the length of the road between its two endpoints, or the amount of time required to travel from one endpoint to the other, or the cost to bury cable along the road from one end to the other. In such cases, instead of being interested in just any spanning tree, we may be interested in a **least cost spanning tree**, that is, a spanning tree such that the sum of the costs of the edges of the tree is as small as possible. For example, this would be the least expensive way to connect a set of towns by a communication network, burying the cable in such a way as to minimize the total cost of laying the cable.

This problem is one that can be solved by a **greedy algorithm**. Roughly speaking, a greedy algorithm is one that makes choices that are optimal in the short run. Typically this strategy does not result in an optimal solution in the long run, but in this case this approach works.

**DEFINITION 5.6.1** A weighted graph is a graph  $G$  together with a cost function  $c: E(G) \rightarrow \mathbb{R}^{>0}$ . If  $H$  is a subgraph of  $G$ , the cost of  $H$  is  $c(H) = \sum_{e \in E(H)} c(e)$ .  $\square$

**The Jarník Algorithm.** Given a weighted connected graph  $G$ , we construct a minimum cost spanning tree  $T$  as follows. Choose any vertex  $v_0$  in  $G$  and include it in  $T$ . If vertices  $S = \{v_0, v_1, \dots, v_k\}$  have been chosen, choose an edge with one endpoint in  $S$  and one endpoint not in  $S$  and with smallest weight among all such edges. Let  $v_{k+1}$  be the endpoint of this edge not in  $S$ , and add it and the associated edge to  $T$ . Continue until all vertices of  $G$  are in  $T$ .

This algorithm was discovered by Vojtěch Jarník in 1930, and rediscovered independently by Robert C. Prim in 1957 and Edsger Dijkstra in 1959. It is often called Prim's Algorithm.

The algorithm proceeds by constructing a sequence of trees  $T_1, T_2, \dots, T_{n-1}$ , with  $T_{n-1}$  a spanning tree for  $G$ . At each step, the algorithm adds an edge that will make  $c(T_{i+1})$  as small as possible among all trees that consist of  $T_i$  plus one edge. This is the best choice in the short run, but it is not obvious that in the long run, that is, by the time  $T_{n-1}$  is constructed, that this will turn out to have been the best choice.

**THEOREM 5.6.2** The Jarník Algorithm produces a minimum cost spanning tree.

**Proof.** Suppose  $G$  is connected on  $n$  vertices. Let  $T$  be the spanning tree produced by the algorithm, and  $T_m$  a minimum cost spanning tree. We prove that  $c(T) = c(T_m)$ .

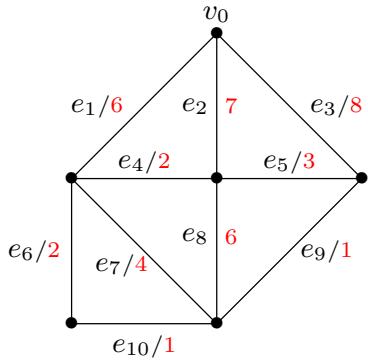
Let  $e_1, e_2, \dots, e_{n-1}$  be the edges of  $T$  in the order in which they were added to  $T$ ; one endpoint of  $e_i$  is  $v_i$ , the other is in  $\{v_0, \dots, v_{i-1}\}$ . We form a sequence of trees  $T_m = T_0, T_1, \dots, T_{n-1} = T$  such that for each  $i$ ,  $c(T_i) = c(T_{i+1})$ , and we conclude that  $c(T_m) = c(T)$ .

If  $e_1$  is in  $T_0$ , let  $T_1 = T_0$ . Otherwise, add edge  $e_1$  to  $T_0$ . This creates a cycle containing  $e_1$  and another edge incident at  $v_0$ , say  $f_1$ . Remove  $f_1$  to form  $T_1$ . Since the algorithm added edge  $e_1$ ,  $c(e_1) \leq c(f_1)$ . If  $c(e_1) < c(f_1)$ , then  $c(T_1) < c(T_0) = c(T_m)$ , a contradiction, so  $c(e_1) = c(f_1)$  and  $c(T_1) = c(T_0)$ .

Suppose we have constructed tree  $T_i$ . If  $e_{i+1}$  is in  $T_i$ , let  $T_{i+1} = T_i$ . Otherwise, add edge  $e_{i+1}$  to  $T_i$ . This creates a cycle, one of whose edges, call it  $f_{i+1}$ , is not in  $e_1, e_2, \dots, e_i$  and has exactly one endpoint in  $\{v_0, \dots, v_i\}$ . Remove  $f_{i+1}$  to create  $T_{i+1}$ . Since the algorithm added  $e_{i+1}$ ,  $c(e_{i+1}) \leq c(f_{i+1})$ . If  $c(e_{i+1}) < c(f_{i+1})$ , then  $c(T_{i+1}) < c(T_i) = c(T_m)$ , a contradiction, so  $c(e_{i+1}) = c(f_{i+1})$  and  $c(T_{i+1}) = c(T_i)$ . ■

### Exercises 5.6.

1. Kruskal's Algorithm is also a greedy algorithm that produces a minimum cost spanning tree for a connected graph  $G$ . Begin by choosing an edge in  $G$  of smallest cost. Assuming that edges  $e_1, e_2, \dots, e_i$  have been chosen, pick an edge  $e_{i+1}$  that does not form a cycle together with  $e_1, e_2, \dots, e_i$  and that has smallest cost among all such edges. The edges  $e_1, e_2, \dots, e_{n-1}$  form a spanning tree for  $G$ . Prove that this spanning tree has minimum cost.
2. Prove that if the edge costs of  $G$  are distinct, there is exactly one minimum cost spanning tree. Give an example of a graph  $G$  with more than one minimum cost spanning tree.
3. In both the Jarník and Kruskal algorithms, it may be that two or more edges can be added at any particular step, and some method is required to choose one over the other. For the graph below, use both algorithms to find a minimum cost spanning tree. Using the labels  $e_i$  on the graph, at each stage pick the edge  $e_i$  that the algorithm specifies and that has the lowest possible  $i$  among all edges available. For the Jarník algorithm, use the designated  $v_0$  as the starting vertex. For each algorithm, list the edges in the order in which they are added. The edge weights  $e_1, e_2, \dots, e_{10}$  are 6, 7, 8, 2, 3, 2, 4, 6, 1, 1, shown in red.



## 5.7 CONNECTIVITY

We have seen examples of connected graphs and graphs that are not connected. While “not connected” is pretty much a dead end, there is much to be said about “how connected” a connected graph is. The simplest approach is to look at how hard it is to disconnect a graph by removing vertices or edges. We assume that all graphs are simple.

If it is possible to disconnect a graph by removing a single vertex, called a **cutpoint**, we say the graph has connectivity 1. If this is not possible, but it is possible to disconnect the graph by removing two vertices, the graph has connectivity 2.

**DEFINITION 5.7.1** If a graph  $G$  is connected, any set of vertices whose removal disconnects the graph is called a **cutset**.  $G$  has connectivity  $k$  if there is a cutset of size  $k$  but no smaller cutset. If there is no cutset and  $G$  has at least two vertices, we say  $G$  has connectivity  $n - 1$ ; if  $G$  has one vertex, its connectivity is undefined. If  $G$  is not connected, we say it has connectivity 0.  $G$  is  $k$ -connected if the connectivity of  $G$  is at least  $k$ . The connectivity of  $G$  is denoted  $\kappa(G)$ .  $\square$

As you should expect from the definition, there are graphs without a cutset: the complete graphs  $K_n$ . If  $G$  is connected but not a  $K_n$ , it has vertices  $v$  and  $w$  that are not adjacent, so removing the  $n - 2$  other vertices leaves a non-connected graph, and so the connectivity of  $G$  is at most  $n - 2$ . Thus, only the complete graphs have connectivity  $n - 1$ .

We do the same thing for edges:

**DEFINITION 5.7.2** If a graph  $G$  is connected, any set of edges whose removal disconnects the graph is called a **cut**.  $G$  has edge connectivity  $k$  if there is a cut of size  $k$  but no smaller cut; the edge connectivity of a one-vertex graph is undefined.  $G$  is  $k$ -edge-connected if the edge connectivity of  $G$  is at least  $k$ . The edge connectivity is denoted  $\lambda(G)$ .  $\square$

Any connected graph with at least two vertices can be disconnected by removing edges: by removing all edges incident with a single vertex the graph is disconnected. Thus,  $\lambda(G) \leq \delta(G)$ , where  $\delta(G)$  is the minimum degree of any vertex in  $G$ . Note that  $\delta(G) \leq n - 1$ , so  $\lambda(G) \leq n - 1$ .

Removing a vertex also removes all of the edges incident with it, which suggests that  $\kappa(G) \leq \lambda(G)$ . This turns out to be true, though not as easy as you might hope. We write  $G - v$  to mean  $G$  with vertex  $v$  removed, and  $G - \{v_1, v_2, \dots, v_k\}$  to mean  $G$  with all of  $\{v_1, v_2, \dots, v_k\}$  removed, and similarly for edges.

**THEOREM 5.7.3**  $\kappa(G) \leq \lambda(G)$ .

**Proof.** We use induction on  $\lambda = \lambda(G)$ . If  $\lambda = 0$ ,  $G$  is disconnected, so  $\kappa = 0$ . If  $\lambda = 1$ , removal of edge  $e$  with endpoints  $v$  and  $w$  disconnects  $G$ . If  $v$  and  $w$  are the only vertices of  $G$ ,  $G$  is  $K_2$  and has connectivity 1. Otherwise, removal of one of  $v$  and  $w$  disconnects  $G$ , so  $\kappa = 1$ .

As a special case we note that if  $\lambda = n - 1$  then  $\delta = n - 1$ , so  $G$  is  $K_n$  and  $\kappa = n - 1$ .

Now suppose  $n - 1 > \lambda = k > 1$ , and removal of edges  $e_1, e_2, \dots, e_k$  disconnects  $G$ . Remove edge  $e_k$  with endpoints  $v$  and  $w$  to form  $G_1$  with  $\lambda(G_1) = k - 1$ . By the induction hypothesis, there are at most  $k - 1$  vertices  $v_1, v_2, \dots, v_j$  such that  $G_2 = G_1 - \{v_1, v_2, \dots, v_j\}$  is disconnected. Since  $k < n - 1$ ,  $k - 1 \leq n - 3$ , and so  $G_2$  has at least 3 vertices.

If both  $v$  and  $w$  are vertices of  $G_2$ , and if adding  $e_k$  to  $G_2$  produces a connected graph  $G_3$ , then removal of one of  $v$  and  $w$  will disconnect  $G_3$  forming  $G_4$ , and  $G_4 = G - \{v_1, v_2, \dots, v_j, v\}$  or  $G_4 = G - \{v_1, v_2, \dots, v_j, w\}$ , that is, removing at most  $k$  vertices disconnects  $G$ . If  $v$  and  $w$  are vertices of  $G_2$  but adding  $e_k$  does not produce a connected graph, then removing  $v_1, v_2, \dots, v_j$  disconnects  $G$ . Finally, if at least one of  $v$  and  $w$  is not in  $G_2$ , then  $G_2 = G - \{v_1, v_2, \dots, v_j\}$  and the connectivity of  $G$  is less than  $k$ . So in all cases,  $\kappa \leq k$ . ■

Graphs that are 2-connected are particularly important, and the following simple theorem is useful.

**THEOREM 5.7.4** If  $G$  has at least three vertices, the following are equivalent:

1.  $G$  is 2-connected
2.  $G$  is connected and has no cutpoint
3. For all distinct vertices  $u, v, w$  in  $G$  there is a path from  $u$  to  $v$  that does not contain  $w$ .

**Proof.** **1  $\Rightarrow$  3:** Since  $G$  is 2-connected,  $G$  with  $w$  removed is a connected graph  $G'$ . Thus, in  $G'$  there is a path from  $u$  to  $v$ , which in  $G$  is a path from  $u$  to  $v$  avoiding  $w$ .

**3  $\Rightarrow$  2:** If  $G$  has property 3 it is clearly connected. Suppose that  $w$  is a cutpoint, so that  $G' = G - w$  is disconnected. Let  $u$  and  $v$  be vertices in two different components of  $G'$ , so that no path connects them in  $G'$ . Then every path joining  $u$  to  $v$  in  $G$  must use  $w$ , a contradiction.

**2  $\Rightarrow$  1:** Since  $G$  has at least 3 vertices and has no cutpoint, its connectivity is at least 2, so it is 2-connected by definition. ■

There are other nice characterizations of 2-connected graphs.

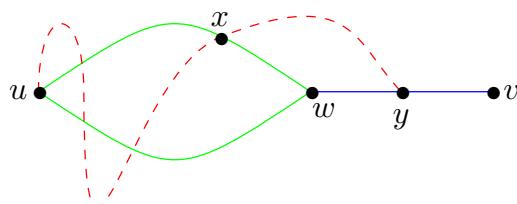
**THEOREM 5.7.5** If  $G$  has at least three vertices, then  $G$  is 2-connected if and only if every two vertices  $u$  and  $v$  are contained in a cycle.

**Proof.**

**if:** Suppose vertex  $w$  is removed from  $G$ , and consider any other vertices  $u$  and  $v$ . In  $G$ ,  $u$  and  $v$  lie on a cycle; even if  $w$  also lies on this cycle, then  $u$  and  $v$  are still connected by a path when  $w$  is removed.

**only if:** Given  $u$  and  $v$  we want to show there is a cycle containing both. Let  $U$  be the set of vertices other than  $u$  that are contained in a cycle with  $u$ . First, we show that  $U$  is non-empty. Let  $w$  be adjacent to  $u$ , and remove the edge  $e$  between them. Since  $\lambda(G) \geq \kappa(G) \geq 2$ ,  $G - e$  is connected. Thus, there is a path from  $u$  to  $w$ ; together with  $e$  this path forms a cycle containing  $u$  and  $w$ , so  $w \in U$ .

For a contradiction, suppose  $v \notin U$ . Let  $w$  be in  $U$  with  $d(w, v) \geq 1$  as small as possible, fix a cycle  $C$  containing  $u$  and  $w$  and a path  $P$  of length  $d(w, v)$  from  $w$  to  $v$ . By the previous theorem, there is a path  $Q$  from  $u$  to  $v$  that does not use  $w$ . Following this path from  $u$ , there is a last vertex  $x$  on the path that is also on the cycle containing  $u$  and  $w$ , and there is a first vertex  $y$  on the path, after  $x$ , with  $y$  also on the path from  $w$  to  $v$  (it is possible that  $y = v$ , but not that  $y = w$ ); see figure 5.7.1. Now starting at  $u$ , proceeding on cycle  $C$  to  $x$  without using  $w$ , then from  $x$  to  $y$  on  $Q$ , then to  $w$  on  $P$ , and finally back to  $u$  on  $C$ , we see that  $y \in U$ . But  $y$  is closer to  $v$  than is  $w$ , a contradiction. Hence  $v \in U$ . ■



**Figure 5.7.1** Point  $y$  closer to  $v$  than  $w$  is a contradiction; path  $Q$  is shown dashed. (See theorem 5.7.5.)

The following corollary is an easy restatement of this theorem.

**COROLLARY 5.7.6** If  $G$  has at least three vertices, then  $G$  is 2-connected if and only if between every two vertices  $u$  and  $v$  there are two **internally disjoint** paths between  $v$  and  $w$ , that is, paths that share only the vertices  $v$  and  $w$ . ■

This version of the theorem suggests a generalization:

**THEOREM 5.7.7 Menger's Theorem** If  $G$  has at least  $k + 1$  vertices, then  $G$  is  $k$ -connected if and only if between every two vertices  $u$  and  $v$  there are  $k$  pairwise internally disjoint paths. □

We first prove Menger's original version of this, a “local” version.

**DEFINITION 5.7.8** If  $v$  and  $w$  are non-adjacent vertices in  $G$ ,  $\kappa_G(v, w)$  is the smallest number of vertices whose removal separates  $v$  from  $w$ , that is, disconnects  $G$  leaving  $v$  and  $w$  in different components. A cutset that separates  $v$  and  $w$  is called a **separating set** for  $v$  and  $w$ .  $p_G(v, w)$  is the maximum number of internally disjoint paths between  $v$  and  $w$ . □

**THEOREM 5.7.9** If  $v$  and  $w$  are non-adjacent vertices in  $G$ ,  $\kappa_G(v, w) = p_G(v, w)$ .

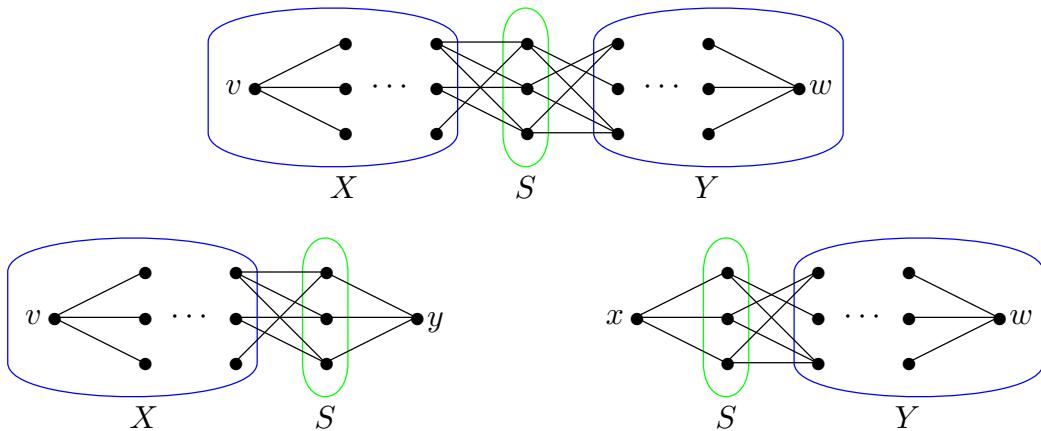
**Proof.** If there are  $k$  internally disjoint paths between  $v$  and  $w$ , then any set of vertices whose removal separates  $v$  from  $w$  must contain at least one vertex from each of the  $k$  paths, so  $\kappa_G(v, w) \geq p_G(v, w)$ .

To finish the proof, we show that there are  $\kappa_G(v, w)$  internally disjoint paths between  $v$  and  $w$ . The proof is by induction on the number of vertices in  $G$ . If  $G$  has two vertices,  $G$  is not connected, and  $\kappa_G(v, w) = p_G(v, w) = 0$ . Now suppose  $G$  has  $n > 2$  vertices and  $\kappa_G(v, w) = k$ .

Note that removal of either  $N(v)$  or  $N(w)$  separates  $v$  from  $w$ , so no separating set  $S$  of size  $k$  can properly contain  $N(v)$  or  $N(w)$ . Now we address two cases:

**Case 1:** Suppose there is a set  $S$  of size  $k$  that separates  $v$  from  $w$ , and  $S$  contains a vertex not in  $N(v)$  or  $N(w)$ .  $G - S$  is disconnected, and one component  $G_1$  contains  $v$ . Since  $S$  does not contain  $N(v)$ ,  $G_1$  has at least two vertices; let  $X = V(G_1)$  and  $Y = V(G) - S - X$ . Since  $S$  does not contain  $N(w)$ ,  $Y$  contains at least two vertices. Now we form two new graphs: Form  $H_X$  by starting with  $G - Y$  and adding a vertex  $y$  adjacent to every vertex of  $S$ . Form  $H_Y$  by starting with  $G - X$  and adding a vertex  $x$  adjacent to every vertex of  $S$ ; see figure 5.7.2. Since  $X$  and  $Y$  each contain at least two vertices,  $H_X$  and  $H_Y$  are smaller than  $G$ , and so the induction hypothesis applies to them.

Clearly  $S$  separates  $v$  from  $y$  in  $H_X$  and  $w$  from  $x$  in  $H_Y$ . Moreover, any set that separates  $v$  from  $y$  in  $H_X$  separates  $v$  from  $w$  in  $G$ , so  $\kappa_{H_X}(v, y) = \kappa_G(v, w) = k$ . Similarly,  $\kappa_{H_Y}(x, w) = \kappa_G(v, w) = k$ . Hence, by the induction hypothesis, there are  $k$  internally disjoint paths from  $v$  to  $y$  in  $H_X$  and  $k$  internally disjoint paths from  $x$  to  $w$  in  $H_Y$ . Each of these paths uses one vertex of  $S$ ; by eliminating  $x$  and  $y$  and joining the paths at the vertices of  $S$ , we produce  $k$  internally disjoint paths from  $v$  to  $w$ .



**Figure 5.7.2** Case 1: Top figure is  $G$ , lower left is  $H_X$ , lower right is  $H_Y$ .

**Case 2:** Now we suppose that any set  $S$  separating  $v$  and  $w$  is a subset of  $N(v) \cup N(w)$ ; pick such an  $S$ . If there is a vertex  $u$  not in  $\{v, w\} \cup N(v) \cup N(w)$ , consider  $G - u$ . This  $u$  is not in any set of size  $k$  that separates  $v$  from  $w$ , for if it were we would be in Case 1. Since  $S$  separates  $v$  from  $w$  in  $G - u$ ,  $\kappa_{G-u}(v, w) \leq k$ . But if some smaller set  $S'$  separates  $v$  from  $w$  in  $G - u$ , then  $S' \cup \{u\}$  separates  $v$  from  $w$  in  $G$ , a contradiction, so  $\kappa_{G-u}(v, w) = k$ . By the induction hypothesis, there are  $k$  internally disjoint paths from  $v$  to  $w$  in  $G - u$  and hence in  $G$ .

We are left with  $V(G) = \{v, w\} \cup N(v) \cup N(w)$ . Suppose there is a vertex  $u$  in  $N(v) \cap N(w)$ . Then  $u$  is in every set that separates  $v$  from  $w$ , so  $\kappa_{G-u} = k - 1$ . By the induction hypothesis, there are  $k - 1$  internally disjoint paths from  $v$  to  $w$  in  $G - u$  and together with the path  $v, u, w$ , they comprise  $k$  internally disjoint paths from  $v$  to  $w$  in  $G$ .

Finally, suppose that  $N(v) \cap N(w) = \emptyset$ . Form a bipartite graph  $B$  with vertices  $N(v) \cup N(w)$  and any edges of  $G$  that have one endpoint in  $N(v)$  and the other in  $N(w)$ . Every set separating  $v$  from  $w$  in  $G$  must include one endpoint of every edge in  $B$ , that is, must be a vertex cover in  $B$ , and conversely, every vertex cover in  $B$  separates  $v$  from  $w$  in  $G$ . Thus, the minimum size of a vertex cover in  $B$  is  $k$ , and so there is a matching in  $B$  of size  $k$ , by theorem 4.5.6. The edges of this matching, together with the edges incident at  $v$  and  $w$ , form  $k$  internally disjoint paths from  $v$  to  $w$  in  $G$ . ■

**Proof of Menger's Theorem (5.7.7).** Suppose first that between every two vertices  $v$  and  $w$  in  $G$  there are  $k$  internally disjoint paths. If  $G$  is not  $k$ -connected, the connectivity of  $G$  is at most  $k - 1$ , and because  $G$  has at least  $k + 1$  vertices, there is a cutset  $S$  of  $G$  with size at most  $k - 1$ . Let  $v$  and  $w$  be vertices in two different components of  $G - S$ ; in  $G$  these vertices are joined by  $k$  internally disjoint paths. Since there is no path from  $v$  to  $w$  in  $G - S$ , each of these  $k$  paths contains a vertex of  $S$ , but this is impossible since  $S$  has size less than  $k$ , and the paths share no vertices other than  $v$  and  $w$ . This contradiction shows that  $G$  is  $k$ -connected.

Now suppose  $G$  is  $k$ -connected.

If  $v$  and  $w$  are not adjacent,  $\kappa_G(v, w) \geq k$  and by the previous theorem there are  $p_G(v, w) = \kappa_G(v, w)$  internally disjoint paths between  $v$  and  $w$ .

If  $v$  and  $w$  are connected by edge  $e$ , consider  $G - e$ . If there is a cutset of  $G - e$  of size less than  $k - 1$ , call it  $S$ , then either  $S \cup \{v\}$  or  $S \cup \{w\}$  is a cutset of  $G$  of size less than  $k$ , a contradiction. (Since  $G$  has at least  $k + 1$  vertices,  $G - S$  has at least three vertices.) Thus,  $\kappa_{G-e}(v, w) \geq k - 1$  and by the previous theorem there are at least  $k - 1$  internally disjoint paths between  $v$  and  $w$  in  $G - e$ . Together with the path  $v, w$  using edge  $e$ , these form  $k$  internally disjoint paths between  $v$  and  $w$  in  $G$ . ■

• • •

We return briefly to 2-connectivity. The next theorem can sometimes be used to provide the induction step in an induction proof.

**THEOREM 5.7.10 The Handle Theorem** Suppose  $G$  is 2-connected and  $K$  is a 2-connected proper subgraph of  $G$ . Then there are subgraphs  $L$  and  $H$  (the handle) of  $G$  such that  $L$  is 2-connected,  $L$  contains  $K$ ,  $H$  is a simple path,  $L$  and  $H$  share exactly the endpoints of  $H$ , and  $G$  is the union of  $L$  and  $H$ .

**Proof.** Given  $G$  and  $K$ , let  $L$  be a maximal proper subgraph of  $G$  containing  $K$ . If  $V(L) = V(G)$ , let  $e$  be an edge not in  $L$ . Since  $L$  plus the edge  $e$  is 2-connected, it must be  $G$ , by the maximality of  $L$ . Hence  $H$  is the path consisting of  $e$  and its endpoints.

Suppose that  $v$  is in  $V(G)$  but not  $V(L)$ . Let  $u$  be a vertex of  $L$ . Since  $G$  is 2-connected, there is a cycle containing  $v$  and  $u$ . Following the cycle from  $v$  to  $u$ , let  $w$  be the first vertex in  $L$ . Continuing on the cycle from  $u$  to  $v$ , let  $x$  be the last vertex in  $L$ . Let  $P$  be the path continuing around the cycle:  $(x, v_1, v_2, \dots, v_k, v = v_{k+1}, v_{k+2}, \dots, v_m, w)$ . If  $x \neq w$ , let  $H = P$ . Since  $L$  together with  $H$  is 2-connected, it is  $G$ , as desired.

If  $x = w$  then  $x = w = u$ . Let  $y$  be a vertex of  $L$  other than  $u$ . Since  $G$  is 2-connected, there is a path  $P_1$  from  $v$  to  $y$  that does not include  $u$ . Let  $v_j$  be the last vertex on  $P_1$  that is in  $\{v_1, \dots, v, \dots, v_m\}$ ; without loss of generality, suppose  $j \geq k + 1$ . Then let  $H$  be the path  $(u, v_1, \dots, v = v_{k+1}, \dots, v_j, \dots, y)$ , where from  $v_j$  to  $y$  we follow path  $P_1$ . Now  $L \cup H$

is a 2-connected subgraph of  $G$ , but it is not  $G$ , as it does not contain the edge  $\{u, v_m\}$ , contradicting the maximality of  $L$ . Thus  $x \neq w$ . ■

A graph that is not connected consists of connected components. In a theorem reminiscent of this, we see that connected graphs that are not 2-connected are constructed from 2-connected subgraphs and bridges.

**DEFINITION 5.7.11** A **block** in a graph  $G$  is a maximal induced subgraph on at least two vertices without a cutpoint. □

As usual, maximal here means that the induced subgraph  $B$  cannot be made larger by adding vertices edges. A block is either a 2-connected induced subgraph or a single edge together with its endpoints. Blocks are useful in large part because of this theorem:

**THEOREM 5.7.12** The blocks of  $G$  partition the edges.

**Proof.** We need to show that every edge is in exactly one block. If an edge is in no 2-connected induced subgraph of  $G$ , then, together with its endpoints, it is itself a block. Thus, every edge is in some block.

Now suppose that  $B_1$  and  $B_2$  are distinct blocks. This implies that neither is a subgraph of the other, by the maximality condition. Hence, the induced subgraph  $G[V(B_1) \cup V(B_2)]$  is larger than either of  $B_1$  and  $B_2$ . Suppose  $B_1$  and  $B_2$  share an edge, so that they share the endpoints of this edge, say  $u$  and  $v$ . Suppose  $w$  is a vertex in  $V(B_1) \cup V(B_2)$ . Since  $B_1 - w$  and  $B_2 - w$  are connected, so is  $G[(V(B_1) \cup V(B_2)) \setminus \{w\}]$ , because either  $u$  or  $v$  is in  $(V(B_1) \cup V(B_2)) \setminus \{w\}$ . Thus  $G[V(B_1) \cup V(B_2)]$  has no cutpoint and so it is 2-connected and strictly contains  $B_1$  and  $B_2$ , contradicting the maximality property of blocks. Thus, every edge is in at most one block. ■

If  $G$  has a single block, it is either  $K_2$  or is 2-connected, and any 2-connected graph has a single block.

**THEOREM 5.7.13** If  $G$  is connected but not 2-connected, then every vertex that is in two blocks is a cutpoint of  $G$ .

**Proof.** Suppose  $w$  is in  $B_1$  and  $B_2$ , but  $G - w$  is connected. Then there is a path  $v_1, v_2, \dots, v_k$  in  $G - w$ , with  $v_1 \in B_1$  and  $v_k \in B_2$ . But then  $G[V(B_1) \cup V(B_2) \cup \{v_1, v_2, \dots, v_k\}]$  is 2-connected and contains both  $B_1$  and  $B_2$ , a contradiction. ■

### Exercises 5.7.

- Suppose a simple graph  $G$  on  $n \geq 2$  vertices has at least  $\frac{(n-1)(n-2)}{2} + 1$  edges. Prove that  $G$  is connected.

2. Suppose a general graph  $G$  has exactly two odd-degree vertices,  $v$  and  $w$ . Let  $G'$  be the graph created by adding an edge joining  $v$  to  $w$ . Prove that  $G'$  is connected if and only if  $G$  is connected.
3. Suppose  $G$  is simple with degree sequence  $d_1 \leq d_2 \leq \dots \leq d_n$ , and for  $k \leq n-d_n-1$ ,  $d_k \geq k$ . Show  $G$  is connected.
4. Recall that a graph is  $k$ -regular if all the vertices have degree  $k$ . What is the smallest integer  $k$  that makes this true?

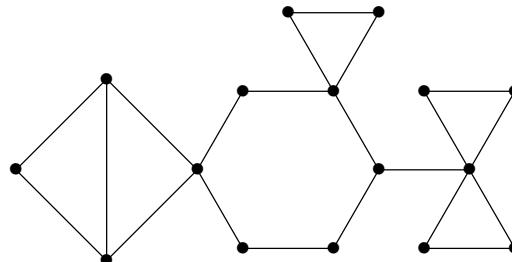
If  $G$  is simple, has  $n$  vertices,  $m \geq k$ , and  $G$  is  $m$ -regular, then  $G$  is connected.

(Of course  $k$  depends on  $n$ .)

5. Suppose  $G$  has at least one edge. Show that  $G$  is 2-connected if and only if for all vertices  $v$  and edges  $e$  there is a cycle containing  $v$  and  $e$ .
6. Find a simple graph with  $\kappa(G) < \lambda(G) < \delta(G)$ .
7. Suppose  $\lambda(G) = k > 0$ . Show that there are sets of vertices  $U$  and  $V$  that partition the vertices of  $G$ , and such that there are exactly  $k$  edges with one endpoint in  $U$  and one endpoint in  $V$ .
8. Find  $\lambda(K_{m,n})$ , where both  $m$  and  $n$  are at least 1. ( $K_{m,n}$  is the complete bipartite graph on  $m$  and  $n$  vertices: the parts have  $m$  and  $n$  vertices, and every pair of vertices, one from each part, is connected by an edge.)
9. Suppose  $G$  is a connected graph. The **block-cutpoint graph** of  $G$ ,  $BC(G)$  is formed as follows: Let vertices  $c_1, c_2, \dots, c_k$  be the cutpoints of  $G$ , and let the blocks of  $G$  be  $B_1, \dots, B_l$ . The vertices of  $BC(G)$  are  $c_1, \dots, c_k, B_1, \dots, B_l$ . Add an edge  $\{B_i, c_j\}$  if and only if  $c_j \in B_i$ . Show that the block-cutpoint graph is a tree.

Note that a cutpoint is contained in at least two blocks, so that all pendant vertices of the block-cutpoint graph are blocks. These blocks are called **endblocks**.

10. Draw the block-cutpoint graph of the graph below.



11. Show that the complement of a disconnected graph is connected. Is the complement of a connected graph always disconnected? (The complement  $\overline{G}$  of graph  $G$  has the same vertices as  $G$ , and  $\{v, w\}$  is an edge of  $\overline{G}$  if and only if it is not an edge of  $G$ .)

## 5.8 GRAPH COLORING

As we briefly discussed in section 1.1, the most famous graph coloring problem is certainly the map coloring problem, proposed in the nineteenth century and finally solved in 1976.

**DEFINITION 5.8.1** A **proper coloring** of a graph is an assignment of colors to the vertices of the graph so that no two adjacent vertices have the same color.  $\square$

Usually we drop the word “proper” unless other types of coloring are also under discussion. Of course, the “colors” don’t have to be actual colors; they can be any distinct labels—integers, for example. If a graph is not connected, each connected component can be colored independently; except where otherwise noted, we assume graphs are connected. We also assume graphs are simple in this section.

Graph coloring has many applications in addition to its intrinsic interest.

**EXAMPLE 5.8.2** If the vertices of a graph represent academic classes, and two vertices are adjacent if the corresponding classes have people in common, then a coloring of the vertices can be used to schedule class meetings. Here the colors would be schedule times, such as 8MWF, 9MWF, 11TTh, etc.  $\square$

**EXAMPLE 5.8.3** If the vertices of a graph represent radio stations, and two vertices are adjacent if the stations are close enough to interfere with each other, a coloring can be used to assign non-interfering frequencies to the stations.  $\square$

**EXAMPLE 5.8.4** If the vertices of a graph represent traffic signals at an intersection, and two vertices are adjacent if the corresponding signals cannot be green at the same time, a coloring can be used to designate sets of signals than can be green at the same time.  $\square$

Graph coloring is closely related to the concept of an **independent set**.

**DEFINITION 5.8.5** A set  $S$  of vertices in a graph is independent if no two vertices of  $S$  are adjacent.  $\square$

If a graph is properly colored, the vertices that are assigned a particular color form an independent set. Given a graph  $G$  it is easy to find a proper coloring: give every vertex a different color. Clearly the interesting quantity is the minimum number of colors required for a coloring. It is also easy to find independent sets: just pick vertices that are mutually non-adjacent. A single vertex set, for example, is independent, and usually finding larger independent sets is easy. The interesting quantity is the maximum size of an independent set.

**DEFINITION 5.8.6** The **chromatic number** of a graph  $G$  is the minimum number of colors required in a proper coloring; it is denoted  $\chi(G)$ . The **independence number** of  $G$  is the maximum size of an independent set; it is denoted  $\alpha(G)$ .  $\square$

The natural first question about these **graphical parameters** is: how small or large can they be in a graph  $G$  with  $n$  vertices. It is easy to see that

$$\begin{aligned} 1 \leq \chi(G) \leq n \\ 1 \leq \alpha(G) \leq n \end{aligned}$$

and that the limits are all attainable: A graph with no edges has chromatic number 1 and independence number  $n$ , while a complete graph has chromatic number  $n$  and independence number 1. These inequalities are thus not very interesting. We will see some that are more interesting.

Another natural question: What is the relation between the chromatic number of a graph  $G$  and chromatic number of a subgraph of  $G$ ? This too is simple, but quite useful at times.

**THEOREM 5.8.7** If  $H$  is a subgraph of  $G$ ,  $\chi(H) \leq \chi(G)$ .

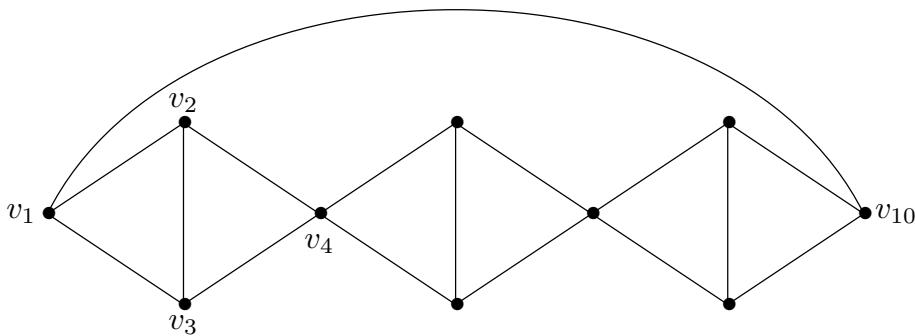
**Proof.** Any coloring of  $G$  provides a proper coloring of  $H$ , simply by assigning the same colors to vertices of  $H$  that they have in  $G$ . This means that  $H$  can be colored with  $\chi(G)$  colors, perhaps even fewer, which is exactly what we want. ■

Often this fact is interesting “in reverse”. For example, if  $G$  has a subgraph  $H$  that is a complete graph  $K_m$ , then  $\chi(H) = m$  and so  $\chi(G) \geq m$ . A subgraph of  $G$  that is a complete graph is called a **clique**, and there is an associated graphical parameter.

**DEFINITION 5.8.8** The **clique number** of a graph  $G$  is the largest  $m$  such that  $K_m$  is a subgraph of  $G$ . □

It is tempting to speculate that the *only* way a graph  $G$  could require  $m$  colors is by having such a subgraph. This is false; graphs can have high chromatic number while having low clique number; see figure 5.8.1. It is easy to see that this graph has  $\chi \geq 3$ , because there are many 3-cliques in the graph. In general it can be difficult to show that a graph cannot be colored with a given number of colors, but in this case it is easy to see that the graph cannot in fact be colored with three colors, because so much is “forced”. Suppose the graph can be colored with 3 colors. Starting at the left if vertex  $v_1$  gets color 1, then  $v_2$  and  $v_3$  must be colored 2 and 3, and vertex  $v_4$  must be color 1. Continuing,  $v_{10}$  must be color 1, but this is not allowed, so  $\chi > 3$ . On the other hand, since  $v_{10}$  can be colored 4, we see  $\chi = 4$ .

Paul Erdős showed in 1959 that there are graphs with arbitrarily large chromatic number and arbitrarily large **girth** (the girth is the size of the smallest cycle in a graph). This is much stronger than the existence of graphs with high chromatic number and low clique number.



**Figure 5.8.1** A graph with clique number 3 and chromatic number 4.

Bipartite graphs with at least one edge have chromatic number 2, since the two parts are each independent sets and can be colored with a single color. Conversely, if a graph can be 2-colored, it is bipartite, since all edges connect vertices of different colors. This means it is easy to identify bipartite graphs: Color any vertex with color 1; color its neighbors color 2; continuing in this way will or will not successfully color the whole graph with 2 colors. If it fails, the graph cannot be 2-colored, since all choices for vertex colors are forced.

If a graph is properly colored, then each **color class** (a color class is the set of all vertices of a single color) is an independent set.

**THEOREM 5.8.9** In any graph  $G$  on  $n$  vertices,  $\frac{n}{\alpha} \leq \chi$ .

**Proof.** Suppose  $G$  is colored with  $\chi$  colors. Since each color class is independent, the size of any color class is at most  $\alpha$ . Let the color classes be  $V_1, V_2, \dots, V_\chi$ . Then

$$n = \sum_{i=1}^{\chi} |V_i| \leq \chi\alpha,$$

as desired. ■

We can improve the upper bound on  $\chi(G)$  as well. In any graph  $G$ ,  $\Delta(G)$  is the maximum degree of any vertex.

**THEOREM 5.8.10** In any graph  $G$ ,  $\chi \leq \Delta + 1$ .

**Proof.** We show that we can always color  $G$  with  $\Delta + 1$  colors by a simple **greedy algorithm**: Pick a vertex  $v_n$ , and list the vertices of  $G$  as  $v_1, v_2, \dots, v_n$  so that if  $i < j$ ,  $d(v_i, v_n) \geq d(v_j, v_n)$ , that is, we list the vertices farthest from  $v_n$  first. We use integers  $1, 2, \dots, \Delta + 1$  as colors. Color  $v_1$  with 1. Then for each  $v_i$  in order, color  $v_i$  with the smallest integer that does not violate the proper coloring requirement, that is, which is

different than the colors already assigned to the neighbors of  $v_i$ . For  $i < n$ , we claim that  $v_i$  is colored with one of  $1, 2, \dots, \Delta$ .

This is certainly true for  $v_1$ . For  $1 < i < n$ ,  $v_i$  has at least one neighbor that is not yet colored, namely, a vertex closer to  $v_n$  on a shortest path from  $v_n$  to  $v_i$ . Thus, the neighbors of  $v_i$  use at most  $\Delta - 1$  colors from the colors  $1, 2, \dots, \Delta$ , leaving at least one color from this list available for  $v_i$ .

Once  $v_1, \dots, v_{n-1}$  have been colored, all neighbors of  $v_n$  have been colored using the colors  $1, 2, \dots, \Delta$ , so color  $\Delta + 1$  may be used to color  $v_n$ . ■

Note that if  $d(v_n) < \Delta$ , even  $v_n$  may be colored with one of the colors  $1, 2, \dots, \Delta$ . Since the choice of  $v_n$  was arbitrary, we may choose  $v_n$  so that  $d(v_n) < \Delta$ , unless all vertices have degree  $\Delta$ , that is, if  $G$  is regular. Thus, we have proved somewhat more than stated, namely, that any graph  $G$  that is not regular has  $\chi \leq \Delta$ . (If instead of choosing the particular order of  $v_1, \dots, v_n$  that we used we were to list them in arbitrary order, even vertices other than  $v_n$  might require use of color  $\Delta + 1$ . This gives a slightly simpler proof of the stated theorem.) We state this as a corollary.

**COROLLARY 5.8.11** If  $G$  is not regular,  $\chi \leq \Delta$ . ■

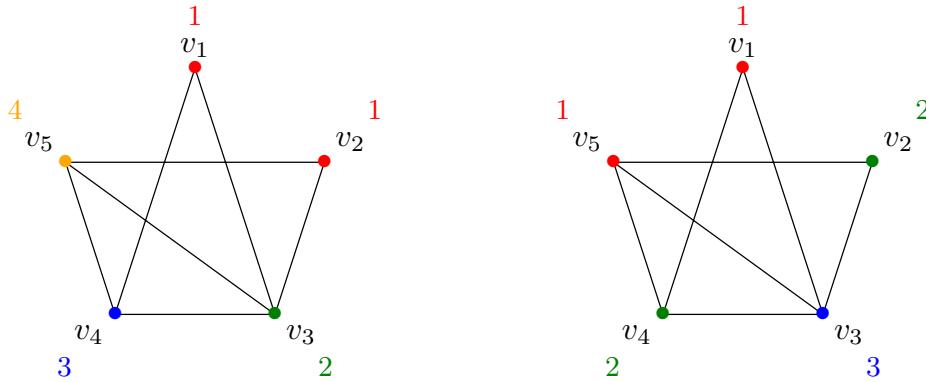
There are graphs for which  $\chi = \Delta + 1$ : any cycle of odd length has  $\Delta = 2$  and  $\chi = 3$ , and  $K_n$  has  $\Delta = n - 1$  and  $\chi = n$ . Of course, these are regular graphs. It turns out that these are the only examples, that is, if  $G$  is not an odd cycle or a complete graph, then  $\chi(G) \leq \Delta(G)$ .

**THEOREM 5.8.12 Brooks's Theorem** If  $G$  is a graph other than  $K_n$  or  $C_{2n+1}$ ,  $\chi \leq \Delta$ . ■

The greedy algorithm will not always color a graph with the smallest possible number of colors. Figure 5.8.2 shows a graph with chromatic number 3, but the greedy algorithm uses 4 colors if the vertices are ordered as shown.

In general, it is difficult to compute  $\chi(G)$ , that is, it takes a large amount of computation, but there is a simple algorithm for graph coloring that is not fast. Suppose that  $v$  and  $w$  are non-adjacent vertices in  $G$ . Denote by  $G + \{v, w\} = G + e$  the graph formed by adding edge  $e = \{v, w\}$  to  $G$ . Denote by  $G/e$  the graph in which  $v$  and  $w$  are “identified”, that is,  $v$  and  $w$  are replaced by a single vertex  $x$  adjacent to all neighbors of  $v$  and  $w$ . (But note that we do not introduce multiple edges: if  $u$  is adjacent to both  $v$  and  $w$  in  $G$ , there will be a single edge from  $x$  to  $u$  in  $G/e$ .)

Consider a proper coloring of  $G$  in which  $v$  and  $w$  are different colors; then this is a proper coloring of  $G + e$  as well. Also, any proper coloring of  $G + e$  is a proper coloring of  $G$  in which  $v$  and  $w$  have different colors. So a coloring of  $G + e$  with the smallest possible



**Figure 5.8.2** A greedy coloring on the left and best coloring on the right.

number of colors is a best coloring of  $G$  in which  $v$  and  $w$  have different colors, that is,  $\chi(G+e)$  is the smallest number of colors needed to color  $G$  so that  $v$  and  $w$  have different colors.

If  $G$  is properly colored and  $v$  and  $w$  have the same color, then this gives a proper coloring of  $G/e$ , by coloring  $x$  in  $G/e$  with the same color used for  $v$  and  $w$  in  $G$ . Also, if  $G/e$  is properly colored, this gives a proper coloring of  $G$  in which  $v$  and  $w$  have the same color, namely, the color of  $x$  in  $G/e$ . Thus,  $\chi(G/e)$  is the smallest number of colors needed to properly color  $G$  so that  $v$  and  $w$  are the same color.

The upshot of these observations is that  $\chi(G) = \min(\chi(G+e), \chi(G/e))$ . This algorithm can be applied recursively, that is, if  $G_1 = G + e$  and  $G_2 = G/e$  then  $\chi(G_1) = \min(\chi(G_1 + e), \chi(G_1/e))$  and  $\chi(G_2) = \min(\chi(G_2 + e), \chi(G_2/e))$ , where of course the edge  $e$  is different in each graph. Continuing in this way, we can eventually compute  $\chi(G)$ , provided that eventually we end up with graphs that are “simple” to color. Roughly speaking, because  $G/e$  has fewer vertices, and  $G + e$  has more edges, we must eventually end up with a complete graph along all branches of the computation. Whenever we encounter a complete graph  $K_m$  it has chromatic number  $m$ , so no further computation is required along the corresponding branch. Let’s make this more precise.

**THEOREM 5.8.13** The algorithm above correctly computes the chromatic number in a finite amount of time.

**Proof.** Suppose that a graph  $G$  has  $n$  vertices and  $m$  edges. The number of pairs of non-adjacent vertices is  $\text{na}(G) = \binom{n}{2} - m$ . The proof is by induction on  $\text{na}$ .

If  $\text{na}(G) = 0$  then  $G$  is a complete graph and the algorithm terminates immediately.

Now we note that  $\text{na}(G+e) < \text{na}(G)$  and  $\text{na}(G/e) < \text{na}(G)$ :

$$\text{na}(G+e) = \binom{n}{2} - (m+1) = \text{na}(G) - 1$$

and

$$\text{na}(G/e) = \binom{n-1}{2} - (m - c),$$

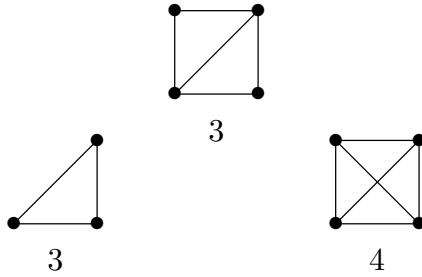
where  $c$  is the number of neighbors that  $v$  and  $w$  have in common. Then

$$\begin{aligned}\text{na}(G/e) &= \binom{n-1}{2} - m + c \\ &\leq \binom{n-1}{2} - m + n - 2 \\ &= \frac{(n-1)(n-2)}{2} - m + n - 2 \\ &= \frac{n(n-1)}{2} - \frac{2(n-1)}{2} - m + n - 2 \\ &= \binom{n}{2} - m - 1 \\ &= \text{na}(G) - 1.\end{aligned}$$

Now if  $\text{na}(G) > 0$ ,  $G$  is not a complete graph, so there are non-adjacent vertices  $v$  and  $w$ . By the induction hypothesis the algorithm computes  $\chi(G + e)$  and  $\chi(G/e)$  correctly, and finally it computes  $\chi(G)$  from these in one additional step. ■

While this algorithm is very inefficient, it is sufficiently fast to be used on small graphs with the aid of a computer.

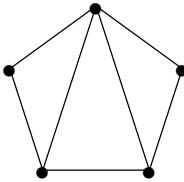
**EXAMPLE 5.8.14** We illustrate with a very simple graph:



The chromatic number of the graph at the top is  $\min(3, 4) = 3$ . (Of course, this is quite easy to see directly.) □

### Exercises 5.8.

- Suppose  $G$  has  $n$  vertices and chromatic number  $k$ . Prove that  $G$  has at least  $\binom{k}{2}$  edges.
- Find the chromatic number of the graph below by using the algorithm in this section. Draw all of the graphs  $G + e$  and  $G/e$  generated by the algorithm in a “tree structure” with the complete graphs at the bottom, label each complete graph with its chromatic number, then propagate the values up to the original graph.



3. Show that  $\chi(G - v)$  is either  $\chi(G)$  or  $\chi(G) - 1$ .
4. Prove theorem 5.8.10 without assuming any particular properties of the order  $v_1, \dots, v_n$ .
5. Prove theorem 5.8.12 as follows. By corollary 5.8.11 we need consider only regular graphs. Regular graphs of degree 2 are easy, so we consider only regular graphs of degree at least 3.

If  $G$  is not 2-connected, show that the blocks of  $G$  may be colored with  $\Delta(G)$  colors, and then the colorings may be altered slightly so that they combine to give a proper coloring of  $G$ .

If  $G$  is 2-connected, show that there are vertices  $u, v, w$  such that  $u$  is adjacent to both  $v$  and  $w$ ,  $v$  and  $w$  are not adjacent, and  $G - v - w$  is connected. Given such vertices, color  $v$  and  $w$  with color 1, then color the remaining vertices by a greedy algorithm similar to that in theorem 5.8.10, with  $u$  playing the role of  $v_n$ .

To show the existence of  $u, v, w$  as required, let  $x$  be a vertex not adjacent to all other vertices. If  $G - x$  is 2-connected, let  $v = x$ , let  $w$  be at distance 2 from  $v$  (justify this), and let a path of length 2 be  $v, u, w$ . Use theorem 5.7.4 to show that  $u, v, w$  have the required properties.

If  $G - x$  is not 2-connected, let  $u = x$  and let  $v$  and  $w$  be (carefully chosen) vertices in two different endblocks of  $G - x$ . Show that  $u, v, w$  have the required properties.

Brooks proved the theorem in 1941; this simpler proof is due to Lovász, 1975.

## 5.9 THE CHROMATIC POLYNOMIAL

We now turn to the number of ways to color a graph  $G$  with  $k$  colors. Of course, if  $k < \chi(G)$ , this is zero. We seek a function  $P_G(k)$  giving the number of ways to color  $G$  with  $k$  colors. Some graphs are easy to do directly.

**EXAMPLE 5.9.1** If  $G$  is  $K_n$ ,  $P_G(k) = k(k - 1)(k - 2) \cdots (k - n + 1)$ , namely, the number of permutations of  $k$  things taken  $n$  at a time. Vertex 1 may be colored any of the  $k$  colors, vertex 2 any of the remaining  $k - 1$  colors, and so on. Note that when  $k < n$ ,  $P_G(k) = 0$ . By exercise 5 in section 1.8, we may also write  $P_G(k) = \sum_{i=0}^n s(n, i)k^i$ .  $\square$

**EXAMPLE 5.9.2** If  $G$  has  $n$  vertices and no edges,  $P_G(k) = k^n$ .  $\square$

Given  $P_G$  it is not hard to compute  $\chi(G)$ ; for example, we could simply plug in the numbers  $1, 2, 3, \dots$  for  $k$  until  $P_G(k)$  is non-zero. This suggests it will be difficult (that is, time consuming) to compute  $P_G$ . We can provide an easy mechanical procedure for the computation, quite similar to the algorithm we presented for computing  $\chi(G)$ .

Suppose  $G$  has edge  $e = \{v, w\}$ , and consider  $P_{G-e}(k)$ , the number of ways to color  $G - e$  with  $k$  colors. Some of the colorings of  $G - e$  are also colorings of  $G$ , but some are not, namely, those in which  $v$  and  $w$  have the same color. How many of these are there? From our discussion of the algorithm for  $\chi(G)$  we know this is the number of colorings of  $G/e$ . Thus,

$$P_G(k) = P_{G-e}(k) - P_{G/e}(k).$$

Since  $G - e$  and  $G/e$  both have fewer edges than  $G$ , we can compute  $P_G$  by applying this formula recursively. Ultimately, we need only compute  $P_G$  for graphs with no edges, which is easy, as in example 5.9.2.

Since  $P_G(k) = k^n$  when  $G$  has no edges, it is then easy to see, and to prove by induction, that  $P_G$  is a polynomial.

**THEOREM 5.9.3** For all  $G$  on  $n$  vertices,  $P_G$  is a polynomial of degree  $n$ , and  $P_G$  is called the **chromatic polynomial** of  $G$ .

**Proof.** The proof is by induction on the number of edges in  $G$ . When  $G$  has no edges, this is example 5.9.2.

Otherwise, by the induction hypothesis,  $P_{G-e}$  is a polynomial of degree  $n$  and  $P_{G/e}$  is a polynomial of degree  $n-1$ , so  $P_G = P_{G-e} - P_{G/e}$  is a polynomial of degree  $n$ . ■

The chromatic polynomial of a graph has a number of interesting and useful properties, some of which are explored in the exercises.

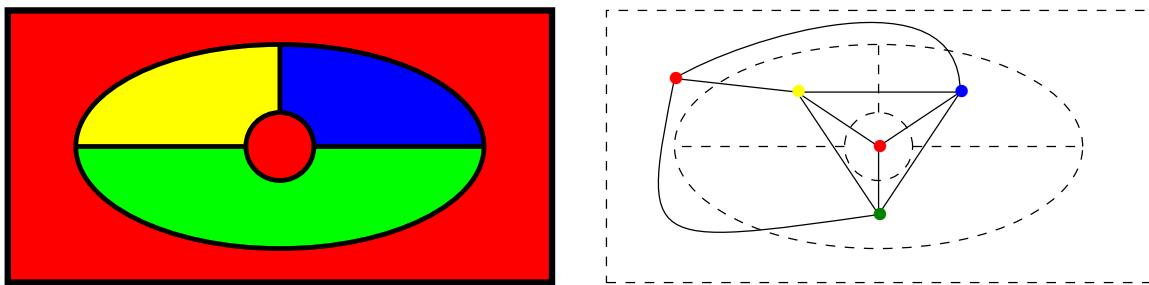
### Exercises 5.9.

1. Show that the leading coefficient of  $P_G$  is 1.
2. Suppose that  $G$  is not connected and has components  $C_1, \dots, C_k$ . Show that  $P_G = \prod_{i=1}^k P_{C_i}$ .
3. Show that the constant term of  $P_G(k)$  is 0. Show that the coefficient of  $k$  in  $P_G(k)$  is non-zero if and only if  $G$  is connected.
4. Show that the coefficient of  $k^{n-1}$  in  $P_G$  is  $-1$  times the number of edges in  $G$ .
5. Show that  $G$  is a tree if and only if  $P_G(k) = k(k-1)^{n-1}$ .
6. Find the chromatic polynomial of  $K_n$  with one edge removed.

## 5.10 COLORING PLANAR GRAPHS

Now we return to the original graph coloring problem: coloring maps. As indicated in section 1.1, the map coloring problem can be turned into a graph coloring problem. Figure 5.10.1 shows the example from section 1.1.

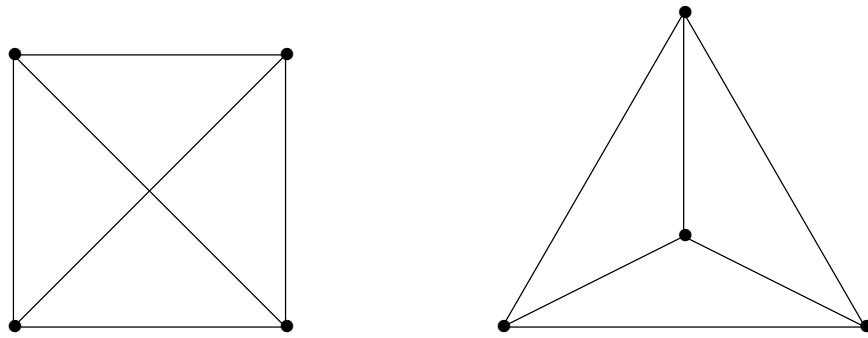
Graphs formed from maps in this way have an important property: they are **planar**.



**Figure 5.10.1** A map and its corresponding graph.

**DEFINITION 5.10.1** A graph  $G$  is planar if it can be represented by a drawing in the plane so that no edges cross.  $\square$

Note that this definition only requires that some representation of the graph has no crossing edges. Figure 5.10.2 shows two representations of  $K_4$ ; since in the second no edges cross,  $K_4$  is planar.



**Figure 5.10.2**  $K_4$  drawn in two ways; the second shows that it is planar.

The number of colors needed to properly color any map is now the number of colors needed to color any planar graph. This problem was first posed in the nineteenth century, and it was quickly conjectured that in all cases four colors suffice. This was finally proved in 1976 (see figure 5.10.3) with the aid of a computer. In 1879, Alfred Kempe gave a proof that was widely known, but was incorrect, though it was not until 1890 that this was noticed by Percy Heawood, who modified the proof to show that five colors suffice to color any planar graph. We will prove this Five Color Theorem, but first we need some other results. We assume all graphs are simple.

**THEOREM 5.10.2 Euler's Formula** Suppose  $G$  is a connected planar graph, drawn so that no edges cross, with  $n$  vertices and  $m$  edges, and that the graph divides the plane



**Figure 5.10.3** The postmark on University of Illinois mail after the Four Color Theorem was proved.

into  $r$  regions. Then

$$r = m - n + 2.$$

**Proof.** The proof is by induction on the number of edges. The base case is  $m = n - 1$ , the minimum number of edges in a connected graph on  $n$  vertices. In this case  $G$  is a tree, and contains no cycles, so the number of regions is 1, and indeed  $1 = (n - 1) - n + 2$ .

Now suppose  $G$  has more than  $n - 1$  edges, so it has a cycle. Remove one edge from a cycle forming  $G'$ , which is connected and has  $r - 1$  regions,  $n$  vertices, and  $m - 1$  edges. By the induction hypothesis  $r - 1 = (m - 1) - n + 2$ , which becomes  $r = m - n + 2$  when we add 1 to each side. ■

**LEMMA 5.10.3** Suppose  $G$  is a simple connected planar graph, drawn so that no edges cross, with  $n \geq 3$  vertices and  $m$  edges, and that the graph divides the plane into  $r$  regions. Then  $m \leq 3n - 6$ .

**Proof.** Let  $f_i$  be the number of edges that adjoin region number  $i$ ; if the same region is on both sides of an edge, that edge is counted twice. We call the edges adjoining a region the boundary edges of the region. Since  $G$  is simple and  $n \geq 3$ , every region is bounded by at least 3 edges. Then  $\sum_{i=1}^r f_i = 2m$ , since each edge is counted twice, once for the region on each side of the edge. From  $r = m - n + 2$  we get  $3r = 3m - 3n + 6$ , and because  $f_i \geq 3$ ,  $3r \leq \sum_{i=1}^r f_i = 2m$ , so  $3m - 3n + 6 \leq 2m$ , or  $m \leq 3n - 6$  as desired. ■

**THEOREM 5.10.4**  $K_5$  is not planar.

**Proof.**  $K_5$  has 5 vertices and 10 edges, and  $10 \not\leq 3 \cdot 5 - 6$ , so by the lemma,  $K_5$  is not planar. ■

**LEMMA 5.10.5** If  $G$  is planar then  $G$  has a vertex of degree at most 5.

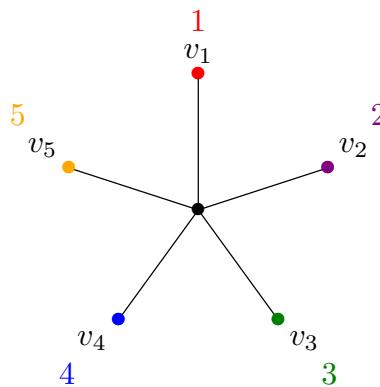
**Proof.** Suppose that  $d(v_i) > 5$  for all  $v_i$ . Then  $2m = \sum_{i=1}^n d(v_i) \geq 6n$ . By lemma 5.10.3,  $3n - 6 \geq m$  so  $6n - 12 \geq 2m$ . Thus  $6n \leq 2m \leq 6n - 12$ , a contradiction. ■

**THEOREM 5.10.6 Five Color Theorem** Every planar graph can be colored with 5 colors.

**Proof.** The proof is by induction on the number of vertices  $n$ ; when  $n \leq 5$  this is trivial.

Now suppose  $G$  is planar on more than 5 vertices; by lemma 5.10.5 some vertex  $v$  has degree at most 5. By the induction hypothesis,  $G - v$  can be colored with 5 colors. Color the vertices of  $G$ , other than  $v$ , as they are colored in a 5-coloring of  $G - v$ . If  $d(v) \leq 4$ , then  $v$  can be colored with one of the 5 colors to give a proper coloring of  $G$  with 5 colors. So we now suppose  $d(v) = 5$ . If the five neighbors of  $v$  are colored with four or fewer of the colors, then again  $v$  can be colored to give a proper coloring of  $G$  with 5 colors.

Now we suppose that all five neighbors of  $v$  have a different color, as indicated in figure 5.10.4.



**Figure 5.10.4** Five neighbors of  $v$  colored with 5 colors:  $v_1$  is red,  $v_2$  is purple,  $v_3$  is green,  $v_4$  is blue,  $v_5$  is orange.

Suppose that in  $G$  there is a path from  $v_1$  to  $v_3$ , and that the vertices along this path are alternately colored red and green; call such a path a red-green alternating path. Then together with  $v$ , this path makes a cycle with  $v_2$  on the inside and  $v_4$  on the outside, or vice versa. This means there cannot be a purple-blue alternating path from  $v_2$  to  $v_4$ . Supposing that  $v_2$  is inside the cycle, we change the colors of all vertices inside the cycle colored purple to blue, and all blue vertices are recolored purple. This is still a proper coloring of all vertices of  $G$  except  $v$ , and now no neighbor of  $v$  is purple, so by coloring  $v$  purple we obtain a proper coloring of  $G$ .

If there is no red-green alternating path from  $v_1$  to  $v_3$ , then we recolor vertices as follows: Change the color of  $v_1$  to green. Change all green neighbors of  $v_1$  to red. Continue to change the colors of vertices from red to green or green to red until there are no conflicts, that is, until a new proper coloring is obtained. Because there is no red-green alternating path from  $v_1$  to  $v_3$ , the color of  $v_3$  will not change. Now no neighbor of  $v$  is colored red, so by coloring  $v$  red we obtain a proper coloring of  $G$ . ■

### Exercises 5.10.

1. Show  $K_{3,3}$  is not planar. (Prove a lemma like lemma 5.10.3 for bipartite graphs, then do something like the proof of theorem 5.10.4.) What is the chromatic number of  $K_{3,3}$ ?

## 5.11 DIRECTED GRAPHS

A **directed graph**, also called a **digraph**, is a graph in which the edges have a direction. This is usually indicated with an arrow on the edge; more formally, if  $v$  and  $w$  are vertices, an edge is an unordered pair  $\{v, w\}$ , while a directed edge, called an **arc**, is an ordered pair  $(v, w)$  or  $(w, v)$ . The arc  $(v, w)$  is drawn as an arrow from  $v$  to  $w$ . If a graph contains both arcs  $(v, w)$  and  $(w, v)$ , this is not a “multiple edge”, as the arcs are distinct. It is possible to have multiple arcs, namely, an arc  $(v, w)$  may be included multiple times in the multiset of arcs. As before, a digraph is called simple if there are no loops or multiple arcs.

We denote by  $E_v^-$  the set of all arcs of the form  $(w, v)$ , and by  $E_v^+$  the set of arcs of the form  $(v, w)$ . The **indegree** of  $v$ , denoted  $d^-(v)$ , is the number of arcs in  $E_v^-$ , and the **outdegree**,  $d^+(v)$ , is the number of arcs in  $E_v^+$ . If the vertices are  $v_1, v_2, \dots, v_n$ , the degrees are usually denoted  $d_1^-, d_2^-, \dots, d_n^-$  and  $d_1^+, d_2^+, \dots, d_n^+$ . Note that both  $\sum_{i=0}^n d_i^-$  and  $\sum_{i=0}^n d_i^+$  count the number of arcs exactly once, and of course  $\sum_{i=0}^n d_i^- = \sum_{i=0}^n d_i^+$ . A **walk** in a digraph is a sequence  $v_1, e_1, v_2, e_2, \dots, v_{k-1}, e_{k-1}, v_k$  such that  $e_k = (v_i, v_{i+1})$ ; if  $v_1 = v_k$ , it is a closed walk or a circuit. A **path** in a digraph is a walk in which all vertices are distinct. It is not hard to show that, as for graphs, if there is a walk from  $v$  to  $w$  then there is a path from  $v$  to  $w$ .

Many of the topics we have considered for graphs have analogues in digraphs, but there are many new topics as well. We will look at one particularly important result in the latter category.

**DEFINITION 5.11.1** A **network** is a digraph with a designated **source**  $s$  and **target**  $t \neq s$ . In addition, each arc  $e$  has a positive capacity,  $c(e)$ .  $\square$

Networks can be used to model transport through a physical network, of a physical quantity like oil or electricity, or of something more abstract, like information.

**DEFINITION 5.11.2** A **flow** in a network is a function  $f$  from the arcs of the digraph to  $\mathbb{R}$ , with  $0 \leq f(e) \leq c(e)$  for all  $e$ , and such that

$$\sum_{e \in E_v^+} f(e) = \sum_{e \in E_v^-} f(e),$$

for all  $v$  other than  $s$  and  $t$ .  $\square$

We wish to assign a value to a flow, equal to the net flow out of the source. Since the substance being transported cannot “collect” or “originate” at any vertex other than  $s$  and  $t$ , it seems reasonable that this value should also be the net flow into the target.

Before we prove this, we introduce some new notation. Suppose that  $U$  is a set of vertices in a network, with  $s \in U$  and  $t \notin U$ . Let  $\overrightarrow{U}$  be the set of arcs  $(v, w)$  with  $v \in U$ ,  $w \notin U$ , and  $\overleftarrow{U}$  be the set of arcs  $(v, w)$  with  $v \notin U$ ,  $w \in U$ .

**THEOREM 5.11.3** For any flow  $f$  in a network, the net flow out of the source is equal to the net flow into the target, namely,

$$\sum_{e \in E_s^+} f(e) - \sum_{e \in E_s^-} f(e) = \sum_{e \in E_t^-} f(e) - \sum_{e \in E_t^+} f(e).$$

**Proof.** We will show first that for any  $U$  with  $s \in U$  and  $t \notin U$ ,

$$\sum_{e \in E_s^+} f(e) - \sum_{e \in E_s^-} f(e) = \sum_{e \in \overrightarrow{U}} f(e) - \sum_{e \in \overleftarrow{U}} f(e).$$

Consider the following:

$$S = \sum_{v \in U} \left( \sum_{e \in E_v^+} f(e) - \sum_{e \in E_v^-} f(e) \right).$$

The quantity

$$\sum_{e \in E_v^+} f(e) - \sum_{e \in E_v^-} f(e)$$

is zero except when  $v = s$ , by the definition of a flow. Thus, the entire sum  $S$  has value

$$\sum_{e \in E_s^+} f(e) - \sum_{e \in E_s^-} f(e).$$

On the other hand, we can write the sum  $S$  as

$$\sum_{v \in U} \sum_{e \in E_v^+} f(e) - \sum_{v \in U} \sum_{e \in E_v^-} f(e).$$

Every arc  $e = (x, y)$  with both  $x$  and  $y$  in  $U$  appears in both sums, that is, in

$$\sum_{v \in U} \sum_{e \in E_v^+} f(e),$$

when  $v = x$ , and in

$$\sum_{v \in U} \sum_{e \in E_v^-} f(e),$$

when  $v = y$ , and so the flow in such arcs contributes 0 to the overall value. Thus, only arcs with exactly one endpoint in  $U$  make a non-zero contribution, so the entire sum reduces

to

$$\sum_{e \in \overrightarrow{U}} f(e) - \sum_{e \in \overleftarrow{U}} f(e).$$

Thus

$$\sum_{e \in E_s^+} f(e) - \sum_{e \in E_s^-} f(e) = S = \sum_{e \in \overrightarrow{U}} f(e) - \sum_{e \in \overleftarrow{U}} f(e),$$

as desired.

Now let  $U$  consist of all vertices except  $t$ . Then

$$\sum_{e \in E_s^+} f(e) - \sum_{e \in E_s^-} f(e) = \sum_{e \in \overrightarrow{U}} f(e) - \sum_{e \in \overleftarrow{U}} f(e) = \sum_{e \in E_t^-} f(e) - \sum_{e \in E_t^+} f(e),$$

finishing the proof. ■

**DEFINITION 5.11.4** The **value** of a flow, denoted  $\text{val}(f)$ , is  $\sum_{e \in E_s^+} f(e) - \sum_{e \in E_s^-} f(e)$ . A **maximum flow** in a network is any flow  $f$  whose value is the maximum among all flows. □

We next seek to formalize the notion of a “bottleneck”, with the goal of showing that the maximum flow is equal to the amount that can pass through the smallest bottleneck.

**DEFINITION 5.11.5** A **cut** in a network is a set  $C$  of arcs with the property that every path from  $s$  to  $t$  uses an arc in  $C$ , that is, if the arcs in  $C$  are removed from the network there is no path from  $s$  to  $t$ . The capacity of a cut, denoted  $c(C)$ , is

$$\sum_{e \in C} c(e).$$

A minimum cut is one with minimum capacity. A cut  $C$  is minimal if no cut is properly contained in  $C$ . □

Note that a minimum cut is a minimal cut. Clearly, if  $U$  is a set of vertices containing  $s$  but not  $t$ , then  $\overrightarrow{U}$  is a cut.

**LEMMA 5.11.6** Suppose  $C$  is a minimal cut. Then there is a set  $U$  containing  $s$  but not  $t$  such that  $C = \overrightarrow{U}$ .

**Proof.** Let  $U$  be the set of vertices  $v$  such that there is a path from  $s$  to  $v$  using no arc in  $C$ .

Suppose that  $e = (v, w) \in C$ . Since  $C$  is minimal, there is a path  $P$  from  $s$  to  $t$  using  $e$  but no other arc in  $C$ . Thus, there is a path from  $s$  to  $v$  using no arc of  $C$ , so  $v \in U$ . If

there is a path from  $s$  to  $w$  using no arc of  $C$ , then this path followed by the portion of  $P$  that begins with  $w$  is a walk from  $s$  to  $t$  using no arc in  $C$ . This implies there is a path from  $s$  to  $t$  using no arc in  $C$ , a contradiction. Thus  $w \notin U$  and so  $e \in \vec{U}$ . Hence,  $C \subseteq \vec{U}$ .

Suppose that  $e = (v, w) \in \vec{U}$ . Then  $v \in U$  and  $w \notin U$ , so every path from  $s$  to  $w$  uses an arc in  $C$ . Since  $v \in U$ , there is a path from  $s$  to  $v$  using no arc of  $C$ , and this path followed by  $e$  is a path from  $s$  to  $w$ . Hence the arc  $e$  must be in  $C$ , so  $\vec{U} \subseteq C$ .

We have now shown that  $C = \vec{U}$ . ■

Now we can prove a version of the important max-flow, min cut theorem.

**THEOREM 5.11.7** Suppose in a network all arc capacities are integers. Then the value of a maximum flow is equal to the capacity of a minimum cut. Moreover, there is a maximum flow  $f$  for which all  $f(e)$  are integers.

**Proof.** First we show that for any flow  $f$  and cut  $C$ ,  $\text{val}(f) \leq c(C)$ . It suffices to show this for a minimum cut  $C$ , and by lemma 5.11.6 we know that  $C = \vec{U}$  for some  $U$ . Using the proof of theorem 5.11.3 we have:

$$\text{val}(f) = \sum_{e \in \vec{U}} f(e) - \sum_{e \in \vec{U}} f(e) \leq \sum_{e \in \vec{U}} f(e) \leq \sum_{e \in \vec{U}} c(e) = c(\vec{U}).$$

Now if we find a flow  $f$  and cut  $C$  with  $\text{val}(f) = c(C)$ , it follows that  $f$  is a maximum flow and  $C$  is a minimum cut. We present an algorithm that will produce such an  $f$  and  $C$ .

Given a flow  $f$ , which may initially be the zero flow,  $f(e) = 0$  for all arcs  $e$ , do the following:

0. Let  $U = \{s\}$ .

Repeat the next two steps until no new vertices are added to  $U$ .

1. If there is an arc  $e = (v, w)$  with  $v \in U$  and  $w \notin U$ , and  $f(e) < c(e)$ , add  $w$  to  $U$ .
2. If there is an arc  $e = (v, w)$  with  $v \notin U$  and  $w \in U$ , and  $f(e) > 0$ , add  $v$  to  $U$ .

When this terminates, either  $t \in U$  or  $t \notin U$ . If  $t \in U$ , there is a sequence of distinct vertices  $s = v_1, v_2, v_3, \dots, v_k = t$  such that for each  $i$ ,  $1 \leq i < k$ , either  $e = (v_i, v_{i+1})$  is an arc with  $f(e) < c(e)$  or  $e = (v_{i+1}, v_i)$  is an arc with  $f(e) > 0$ . Update the flow by adding 1 to  $f(e)$  for each of the former, and subtracting 1 from  $f(e)$  for each of the latter. This new flow  $f'$  is still a flow: In the first case, since  $f(e) < c(e)$ ,  $f'(e) \leq c(e)$ , and in the second case, since  $f(e) > 0$ ,  $f'(e) \geq 0$ . It is straightforward to check that for each vertex  $v_i$ ,  $1 < i < k$ , that

$$\sum_{e \in E_{v_i}^+} f'(e) = \sum_{e \in E_{v_i}^-} f'(e).$$

In addition,  $\text{val}(f') = \text{val}(f) + 1$ . Now rename  $f'$  to  $f$  and repeat the algorithm.

Eventually, the algorithm terminates with  $t \notin U$  and flow  $f$ . This implies that for each  $e = (v, w)$  with  $v \in U$  and  $w \notin U$ ,  $f(e) = c(e)$ , and for each  $e = (v, w)$  with  $v \notin U$  and  $w \in U$ ,  $f(e) = 0$ . The capacity of the cut  $\overrightarrow{U}$  is

$$\sum_{e \in \overrightarrow{U}} c(e).$$

The value of the flow  $f$  is

$$\sum_{e \in \overrightarrow{U}} f(e) - \sum_{e \in \overleftarrow{U}} f(e) = \sum_{e \in \overrightarrow{U}} c(e) - \sum_{e \in \overleftarrow{U}} 0 = \sum_{e \in \overrightarrow{U}} c(e).$$

Thus we have found a flow  $f$  and cut  $\overrightarrow{U}$  such that

$$\text{val}(f) = c(\overrightarrow{U}),$$

as desired. ■

The max-flow, min-cut theorem is true when the capacities are any positive real numbers, though of course the maximum value of a flow will not necessarily be an integer in this case. It is somewhat more difficult to prove, requiring a proof involving limits.

We have already proved that in a bipartite graph, the size of a maximum matching is equal to the size of a minimum vertex cover, theorem 4.5.6. This turns out to be essentially a special case of the max-flow, min-cut theorem.

**COROLLARY 5.11.8** In a bipartite graph  $G$ , the size of a maximum matching is the same as the size of a minimum vertex cover.

**Proof.** Suppose the parts of  $G$  are  $X = \{x_1, x_2, \dots, x_k\}$  and  $Y = \{y_1, y_2, \dots, y_l\}$ . Create a network as follows: introduce two new vertices  $s$  and  $t$  and arcs  $(s, x_i)$  for all  $i$  and  $(y_i, t)$  for all  $i$ . For each edge  $\{x_i, y_j\}$  in  $G$ , let  $(x_i, y_j)$  be an arc. Let  $c(e) = 1$  for all arcs  $e$ . Now the value of a maximum flow is equal to the capacity of a minimum cut.

Let  $C$  be a minimum cut. If  $(x_i, y_j)$  is an arc of  $C$ , replace it by arc  $(s, x_i)$ . This is still a cut, since any path from  $s$  to  $t$  including  $(x_i, y_j)$  must include  $(s, x_i)$ . Thus, we may suppose that  $C$  contains only arcs of the form  $(s, x_i)$  and  $(y_i, t)$ . Now it is easy to see that

$$K = \{x_i | (s, x_i) \in C\} \cup \{y_i | (y_i, t) \in C\}$$

is a vertex cover of  $G$  with the same size as  $C$ .

Let  $f$  be a maximum flow such that  $f(e)$  is an integer for all  $e$ , which is possible by the max-flow, min-cut theorem. Consider the set of edges

$$M = \{\{x_i, y_j\} | f((x_i, y_j)) = 1\}.$$

If  $\{x_i, y_j\}$  and  $\{x_i, y_m\}$  are both in this set, then the flow out of vertex  $x_i$  is at least 2, but there is only one arc into  $x_i$ ,  $(s, x_i)$ , with capacity 1, contradicting the definition of a

flow. Likewise, if  $\{x_i, y_j\}$  and  $\{x_m, y_j\}$  are both in this set, then the flow into vertex  $y_j$  is at least 2, but there is only one arc out of  $y_j$ ,  $(y_j, t)$ , with capacity 1, also a contradiction. Thus  $M$  is a matching. Moreover, if  $U = \{s, x_1, \dots, x_k\}$  then the value of the flow is

$$\sum_{e \in \overrightarrow{U}} f(e) - \sum_{e \in \overleftarrow{U}} f(e) = \sum_{e \in \overrightarrow{U}} f(e) = |M| \cdot 1 = |M|.$$

Thus  $|M| = \text{val}(f) = c(C) = |K|$ , so we have found a matching and a vertex cover with the same size. This implies that  $M$  is a maximum matching and  $K$  is a minimum vertex cover. ■

### Exercises 5.11.

1. Connectivity in digraphs turns out to be a little more complicated than connectivity in graphs. A digraph is connected if the underlying graph is connected. (The underlying graph of a digraph is produced by removing the orientation of the arcs to produce edges, that is, replacing each arc  $(v, w)$  by an edge  $\{v, w\}$ . Even if the digraph is simple, the underlying graph may have multiple edges.) A digraph is strongly connected if for every vertices  $v$  and  $w$  there is a walk from  $v$  to  $w$ . Give an example of a digraph that is connected but not strongly connected.
2. A digraph has an Euler circuit if there is a closed walk that uses every arc exactly once. Show that a digraph with no vertices of degree 0 has an Euler circuit if and only if it is connected and  $d^+(v) = d^-(v)$  for all vertices  $v$ .
3. A tournament is an oriented complete graph. That is, it is a digraph on  $n$  vertices, containing exactly one of the arcs  $(v, w)$  and  $(w, v)$  for every pair of vertices. A Hamilton path is a walk that uses every vertex exactly once. Show that every tournament has a Hamilton path.
4. Interpret a tournament as follows: the vertices are players. If  $(v, w)$  is an arc, player  $v$  beat  $w$ . Say that  $v$  is a **champion** if for every other player  $w$ , either  $v$  beat  $w$  or  $v$  beat a player who beat  $w$ . Show that a player with the maximum number of wins is a champion. Find a 5-vertex tournament in which every player is a champion.

# 6

## Pólya–Redfield Counting

We have talked about the number of ways to properly color a graph with  $k$  colors, given by the chromatic polynomial. For example, the chromatic polynomial for the graph in figure 6.0.1 is  $k^4 - 4k^3 + 6k^2 - 3k$ , and  $f(2) = 2$ . The two colorings are shown in the figure, but in an obvious sense they are the same coloring, since one can be turned into the other by simply rotating the graph. We will consider a slightly different sort of coloring problem, in which we count the “truly different” colorings of objects.



**Figure 6.0.1**  $C_4$  drawn as a square, colored in two ways.

Many of the “objects” we color will appear to be graphs, but we will usually be interested in them as geometric objects rather than graphs, and we will not require that adjacent vertices be different colors. This will simplify the problems; counting the number of different proper colorings of graphs can also be done, but it is more complicated.

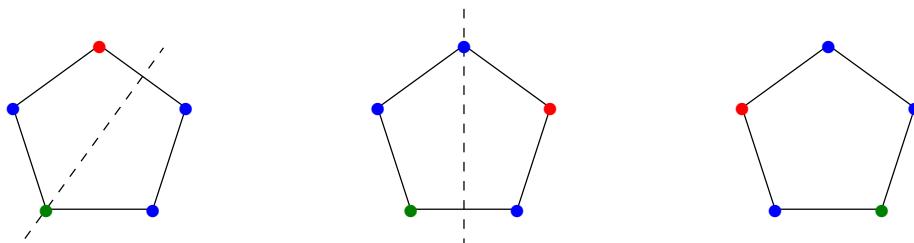
So consider this problem: How many different ways are there to color the vertices of a regular pentagon with  $k$  colors? The number of ways to color the vertices of a fixed pentagon is  $k^5$ , but this includes many duplicates, that is, colorings that are really the same. But what do we mean by “the same” in this case? We might mean that two colorings are the same if we can rotate one to get the other. But what about the colorings in figure 6.0.2? Are they the same? Neither can be rotated to produce the other, but either

can be flipped or reflected through a vertical line to produce the other. In fact we are free to decide what “the same” means, but we will need to make sure that our requirements are consistent.



**Figure 6.0.2** Two colorings of a pentagon.

As an example of what can go wrong if we’re not careful, note that there are five lines through which the pentagon can be reflected onto itself. Suppose we want to consider colorings to be “the same” if one can be produced from the other by a reflection, but not if one can be obtained from the other by rotation. Surely if one coloring can be obtained by two reflections in a row from another, then these colorings should also be the same. But two reflections in a row equal a rotation, as shown in figure 6.0.3. So whenever we have some motions that identify two colorings, we are required to include all combinations of the motions as well. In addition, any time we include a motion, we must include the “inverse” motion. For example, if we say a rotation by  $54^\circ$  degrees produces a coloring that we consider to be the same, a rotation by  $-54^\circ$  must be included as well (we may think of this as a rotation by  $306^\circ$ ). Finally, since any coloring is the same as itself, we must always include the “trivial” motion, namely, doing nothing, or rotation by  $0^\circ$  if you prefer.



**Figure 6.0.3** Two reflections equal a rotation.

## 6.1 GROUPS OF SYMMETRIES

The motions we want to consider can be thought of as permutations, that is, as bijections. For example, the rotation in figure 6.1.1 can be thought of as the function  $\phi$  given by

$$\phi(1) = 3$$

$$\phi(2) = 4$$

$$\phi(3) = 5$$

$$\phi(4) = 1$$

$$\phi(5) = 2,$$

or more compactly we can write this function as  $\begin{pmatrix} 1 & 2 & 3 & 4 & 5 \\ 3 & 4 & 5 & 1 & 2 \end{pmatrix}$ .



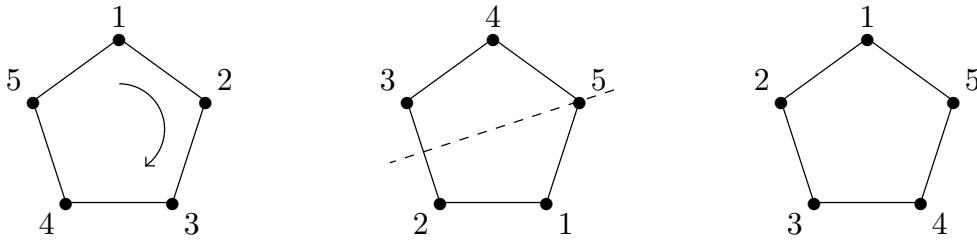
**Figure 6.1.1** The rotation  $\begin{pmatrix} 1 & 2 & 3 & 4 & 5 \\ 3 & 4 & 5 & 1 & 2 \end{pmatrix}$ .

As we would hope, doing two motions in a row corresponds to the composition of the associated functions. For example, the reflection  $\begin{pmatrix} 1 & 2 & 3 & 4 & 5 \\ 3 & 2 & 1 & 5 & 4 \end{pmatrix}$  is shown in figure 6.1.2. Doing first the rotation of figure 6.1.1 and then this reflection is shown in figure 6.1.3, and this does indeed correspond to

$$\begin{pmatrix} 1 & 2 & 3 & 4 & 5 \\ 3 & 2 & 1 & 5 & 4 \end{pmatrix} \circ \begin{pmatrix} 1 & 2 & 3 & 4 & 5 \\ 3 & 4 & 5 & 1 & 2 \end{pmatrix} = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 \\ 1 & 5 & 4 & 3 & 2 \end{pmatrix}.$$



**Figure 6.1.2** The reflection  $\begin{pmatrix} 1 & 2 & 3 & 4 & 5 \\ 3 & 2 & 1 & 5 & 4 \end{pmatrix}$ .



**Figure 6.1.3** The composition  $\begin{pmatrix} 1 & 2 & 3 & 4 & 5 \\ 3 & 2 & 1 & 5 & 4 \end{pmatrix} \circ \begin{pmatrix} 1 & 2 & 3 & 4 & 5 \\ 3 & 4 & 5 & 1 & 2 \end{pmatrix} = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 \\ 1 & 5 & 4 & 3 & 2 \end{pmatrix}$ .

With some restrictions, we may choose any permutations of the vertices as the allowable rearrangements giving colorings that are the same. We have discussed the restrictions in general terms; in terms of permutations we require the following: Suppose that  $G$  is a set of permutations that we wish to use to define the “same coloring” relation. Then the following must be true:

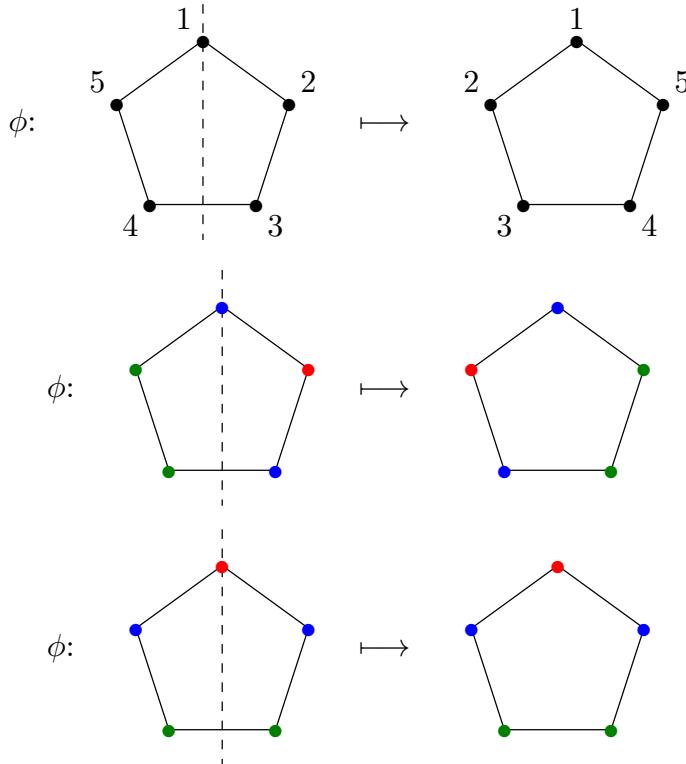
1. If  $\phi$  and  $\sigma$  are in  $G$ , so is  $\phi \circ \sigma$ .
2. The identity permutation,  $\text{id}$ , is in  $G$ .
3. If  $\phi \in G$ ,  $\phi^{-1} \in G$ .

**DEFINITION 6.1.1** If  $G$  has the three properties above it is called a **group** of permutations.  $\square$

**EXAMPLE 6.1.2** The group of all permutations of  $\{1, 2, \dots, n\}$  is denoted  $S_n$ , the **symmetric group** on  $n$  items. It satisfies the three required conditions by simple properties of bijections.  $\square$

In the case of the regular pentagon, there are a number of groups of permutations, but two are of primary interest. The five possible rotations (including the trivial rotation) form a group, the **cyclic group** of size 5. The total number of “rigid motions”, that is, any combination of rotations and reflections that leave the pentagon superimposed on itself, is 10: Once the position of vertex 1 is established, the other vertices can increase from 1 either clockwise or counterclockwise. The rotations provide all of the former, and it is easy to check that the five reflections provide the counterclockwise positions. This is called a **dihedral group** and denoted  $D_5$ .

Suppose that  $G$  is some group of permutations of an object. If  $\phi \in G$ , then  $\phi$  induces a function on the colorings of the object in a natural way, and we can use the same symbol  $\phi$  to represent this function without confusion. If  $c$  is a coloring of the object, then  $\phi(c)$  is the coloring that results by applying  $\phi$  to the colored object, moving the colors with the object. See figure 6.1.4 for examples. We say that  $G$  **acts** on the set of colorings  $C$ .



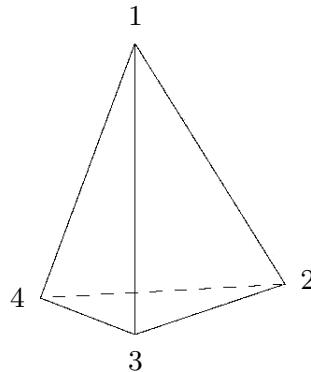
**Figure 6.1.4** Some examples of an induced function on colorings.

If we apply all permutations in  $G$  to a coloring  $c$ , we get all the colorings that we consider to be the same as  $c$  modulo  $G$ . More formally, define  $c_1 \sim c_2$  if there is a  $\phi \in G$  such that  $\phi(c_1) = c_2$ ;  $\sim$  is an equivalence relation on the colorings. The equivalence classes, called **orbits** in this context, group colorings that are the same together. The number of truly different colorings that we want to count is then the number of orbits.

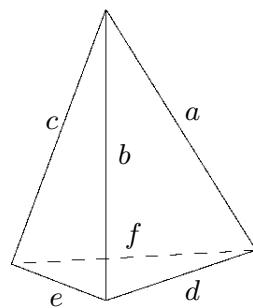
The total number of colorings of the pentagon with  $k$  colors is  $k^5$ . If all orbits were the same size, say  $s$ , then the number of orbits would be  $k^5/s$ . Unfortunately, this is not true. In figure 6.1.4 we see a coloring whose orbit has size at least 3, but the pentagon with all vertices colored red has orbit size 1.

### Exercises 6.1.

- Find the 12 permutations of the vertices of the regular tetrahedron corresponding to the 12 rigid motions of the regular tetrahedron. Use the labeling below.



2. Find the 12 permutations of the edges of the regular tetrahedron corresponding to the 12 rigid motions of the regular tetrahedron. Use the labeling below.



## 6.2 BURNSIDE'S THEOREM

Burnside's Theorem will allow us to count the orbits, that is, the different colorings, in a variety of problems. We first need some lemmas.

If  $c$  is a coloring,  $[c]$  is the orbit of  $c$ , that is, the equivalence class of  $c$ . Let  $G(c)$  be the set of permutations in  $G$  that fix  $c$ , that is, those  $\phi$  such that  $\phi(c) = c$ ; the permutation in figure 6.1.4 fixes the coloring in the bottom row, for example.

**LEMMA 6.2.1**  $G(c)$  is a group of permutations.

**Proof.** We check the properties of a group from definition 6.1.1.

Suppose  $\phi$  and  $\sigma$  both fix  $c$ ; then  $\phi(\sigma(c)) = \phi(c) = c$ , so  $\phi \circ \sigma$  fixes  $c$  and  $\phi \circ \sigma \in G(c)$ .

The identity permutation fixes all colorings, so  $\text{id} \in G(c)$ .

If  $\phi(c) = c$  then  $\phi^{-1}(c) = \phi^{-1}(\phi(c)) = \text{id}(c) = c$ , so  $\phi^{-1} \in G(c)$ . ■

**LEMMA 6.2.2**  $|G| = |[c]| \cdot |G(c)|$ .

**Proof.** For  $\phi$  and  $\sigma$  in  $G$ , define  $\phi \sim \sigma$  if  $\sigma^{-1} \circ \phi \in G(c)$ . This is an equivalence relation:

1.  $\sigma^{-1} \circ \sigma$  is the identity function, which is in  $G(c)$ . Thus  $\sigma \sim \sigma$ , so the relation is reflexive.
2. If  $\phi \sim \sigma$ ,  $\sigma^{-1} \circ \phi \in G(c)$ . Then  $(\sigma^{-1} \circ \phi)^{-1} \in G(c)$ , and  $(\sigma^{-1} \circ \phi)^{-1} = \phi^{-1} \circ \sigma \in G(c)$ , so  $\sigma \sim \phi$  and  $\sim$  is symmetric.

- 3.** If  $\phi \sim \sigma$  and  $\sigma \sim \tau$ , then  $\sigma^{-1} \circ \phi \in G(c)$  and  $\tau^{-1} \circ \sigma \in G(c)$ , so  $(\tau^{-1} \circ \sigma) \circ (\sigma^{-1} \circ \phi) \in G(c)$ . Since  $(\tau^{-1} \circ \sigma) \circ (\sigma^{-1} \circ \phi) = \tau^{-1} \circ \phi$ ,  $\phi \sim \tau$ , and  $\sim$  is transitive.

Now we claim that the equivalence class of  $\phi$  is  $A = \{\phi \circ \sigma \mid \sigma \in G(c)\}$ . First, suppose that  $\sigma \in G(c)$ ; then  $\phi^{-1} \circ \phi \circ \sigma = \sigma \in G(c)$ , so  $\phi \circ \sigma \sim \phi$  and  $A \subseteq [\phi]$ . Next, suppose  $\phi \sim \tau$ , so  $\tau^{-1} \circ \phi = \gamma \in G(c)$ . Then  $\phi \circ \gamma^{-1} = \tau$ , so  $\tau \in A$  and  $[\phi] \subseteq A$ .

Now we show that each equivalence class is the same size as  $G(c)$ . Define  $f: G(c) \rightarrow \{\phi \circ \sigma \mid \sigma \in G(c)\}$  by  $f(\gamma) = \phi \circ \gamma$ . If  $f(\gamma_1) = f(\gamma_2)$ , then

$$\begin{aligned}\phi \circ \gamma_1 &= \phi \circ \gamma_2 \\ \phi^{-1} \circ \phi \circ \gamma_1 &= \phi^{-1} \circ \phi \circ \gamma_2 \\ \gamma_1 &= \gamma_2\end{aligned}$$

so  $f$  is 1–1. Since every  $\phi \circ \gamma \in \{\phi \circ \sigma \mid \sigma \in G(c)\}$  is  $f(\gamma)$ ,  $f$  is onto.

Thus the number of equivalence classes is  $|G|/|G(c)|$ .

Finally, we show that the number of equivalence classes is  $|[c]|$ . Let the set of equivalence classes in  $G$  be  $E$ , that is,  $E = \{[\phi] \mid \phi \in G\}$ . We define  $g: [c] \rightarrow E$  and show that  $g$  is a bijection. Suppose  $d \in [c]$ , so  $d = \sigma(c)$  for some  $\sigma \in G$ . Let  $g(d) = [\sigma]$ .

First, we show  $g$  is well-defined. If  $d = \sigma_1(c) = \sigma_2(c)$ , then  $\sigma_2^{-1} \circ \sigma_1(c) = c$ , so  $\sigma_1 \sim \sigma_2$  and  $[\sigma_1] = [\sigma_2]$ , that is,  $g(\sigma_1(c)) = g(\sigma_2(c))$ .

Next, suppose  $g(d_1) = g(d_2)$ . This means that  $d_1 = \sigma_1(c)$ ,  $d_2 = \sigma_2(c)$ , and  $[\sigma_1] = [\sigma_2]$ . Hence  $\sigma_2^{-1} \circ \sigma_1(c) = c$ , so  $\sigma_1(c) = \sigma_2(c)$  and thus  $d_1 = d_2$ , so  $g$  is 1–1.

Suppose that  $[\sigma] \in E$ . Then  $g(\sigma(c)) = [\sigma]$ , so  $g$  is onto.

Thus we have

$$|[c]| = |E| = \frac{|G|}{|G(c)|}$$

and

$$|G(c)| \cdot |[c]| = |G|$$

as desired. ■

**COROLLARY 6.2.3** If  $c \sim d$  then  $|G(c)| = |G(d)|$ .

**Proof.** Since  $c \sim d$ ,  $[c] = [d]$ , and

$$\begin{aligned}\frac{|G|}{|G(c)|} &= |[c]| = |[d]| = \frac{|G|}{|G(d)|} \\ |G(c)| &= |G(d)|\end{aligned}$$
■

**DEFINITION 6.2.4** If group  $G$  acts on the colorings of an object and  $\sigma \in G$ ,  $\text{fix}(\sigma)$  is the set of colorings that are fixed by  $\sigma$ . □

**THEOREM 6.2.5 Burnside's Theorem** If group  $G$  acts on the colorings of an object, the number of distinct colorings modulo  $G$  is

$$\frac{1}{|G|} \sum_{\sigma \in G} |\text{fix}(\sigma)|.$$

**Proof.** Let  $C$  be the set of all colorings, and let  $\mathcal{O}$  be the set of orbits. Let  $c_1, c_2, \dots, c_k$  be a list of colorings, one in each orbit, so that the orbits are  $[c_1], [c_2], \dots, [c_k]$ . Consider the sum  $\sum_{c \in C} |G(c)|$ :

$$\begin{aligned} \sum_{c \in C} |G(c)| &= \sum_{i=1}^k \sum_{c \in [c_i]} |G(c)| \\ &= \sum_{i=1}^k \sum_{c \in [c_i]} |G(c_i)| \\ &= \sum_{i=1}^k \sum_{c \in [c_i]} \frac{|G|}{|[c_i]|} \\ &= \sum_{i=1}^k |[c_i]| \frac{|G|}{|[c_i]|} \\ &= \sum_{i=1}^k |G| = |G| \sum_{i=1}^k 1 = |G||\mathcal{O}|. \end{aligned}$$

Then

$$|\mathcal{O}| = \frac{1}{|G|} \sum_{c \in C} |G(c)|.$$

This already gives an interesting formula for  $|\mathcal{O}|$ , but it is unwieldy, since the number of colorings is typically quite large; indeed, since we typically want to compute the number of orbits for  $k$  colors, the number of colorings is not a fixed number.

With just a little more work we can fix this problem:

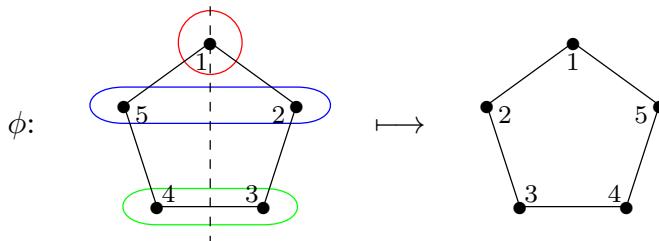
$$\begin{aligned} \sum_{c \in C} |G(c)| &= \sum_{c \in C} \sum_{\sigma \in G(c)} 1 \\ &= \sum_{\sigma \in G} \sum_{c \in \text{fix}(\sigma)} 1 \\ &= \sum_{\sigma \in G} |\text{fix}(\sigma)|. \end{aligned}$$

Now

$$|\mathcal{O}| = \frac{1}{|G|} \sum_{\sigma \in G} |\text{fix}(\sigma)|$$

as desired. ■

Since the group of permutations in a typical problem is fairly small, the sum in Burnside's Theorem is usually manageable. Moreover, we can make the task of computing  $|\text{fix}(\sigma)|$  fairly straightforward. Let's consider a particular example, the permutation of figure 6.1.4, shown again in figure 6.2.1. If we are using  $k$  colors, how many colorings of the pentagon are fixed by this permutation? Since the permutation swaps vertices 2 and 5, they must be the same color if  $\phi$  is to fix the coloring. Likewise vertices 3 and 4 must be the same color; vertex 1 can be any color. Thus, the number of colorings fixed by  $\phi$  is  $k^3$ . It is easy to see that every “flip” permutation of the pentagon is essentially the same, so for each of the five flip permutations, the size of  $\text{fix}(\sigma)$  is  $k^3$ .



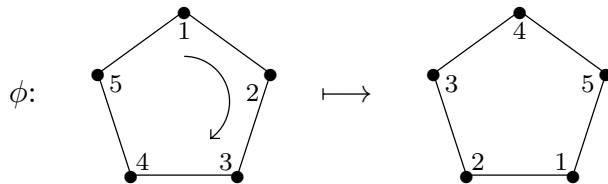
**Figure 6.2.1** The cycles in a vertex permutation.

Every permutation can be written in **cycle form**: The permutation in figure 6.2.1, for example, is  $(1)(2, 5)(3, 4)$ . A cycle in this context is a sequence  $(x_1, x_2, \dots, x_k)$ , meaning that  $\phi(x_1) = x_2$ ,  $\phi(x_2) = x_3$ , and so on until  $\phi(x_k) = x_1$ . Following our reasoning above, the vertices in each cycle must be colored the same color, and the total number of colors fixed by  $\phi$  is  $k^m$ , where  $m$  is the number of cycles.

Let's apply this to another permutation, shown in figure 6.2.2. This permutation consists of a single cycle, so every vertex must have the same color, and the number of colorings fixed by  $\phi$  is  $k^1$ . All rotations of the pentagon consist of a single cycle except the trivial rotation, that is, the identity permutation. In cycle form, the identity permutation is  $(1)(2)(3)(4)(5)$ , so the number of colorings fixed by the identity is  $k^5$ . Putting everything together, we thus have

$$|\mathcal{O}| = \frac{1}{10}(k^5 + k + k + k + k + k^3 + k^3 + k^3 + k^3 + k^3) = \frac{1}{10}(k^5 + 5k^3 + 4k).$$

For example, the number of different 3-colorings is  $(3^5 + 5 \cdot 3^3 + 4 \cdot 3)/10 = 39$ .



**Figure 6.2.2** The permutation  $(1, 3, 5, 2, 4)$  is a single cycle.

**EXAMPLE 6.2.6** We find the number of distinct colorings of the vertices of a square with  $k$  colors, modulo  $D_4$ , the dihedral group of size 8. The elements of  $D_4$  are the four rotations  $r_0, r_{90}, r_{180}, r_{270}$ , where  $r_i$  is the counterclockwise rotation by  $i$  degrees, and the four reflections  $f_H, f_V, f_D, f_A$ , as indicated in figure 6.2.3.



**Figure 6.2.3** The reflection axes for  $f_H, f_V, f_D$ , and  $f_A$ .

In cycle notation these permutations are:

$$r_0 = (1)(2)(3)(4)$$

$$r_{90} = (1, 4, 3, 2)$$

$$r_{180} = (1, 3)(2, 4)$$

$$r_{270} = (1, 2, 3, 4)$$

$$f_H = (1, 4)(2, 3)$$

$$f_V = (1, 2)(3, 4)$$

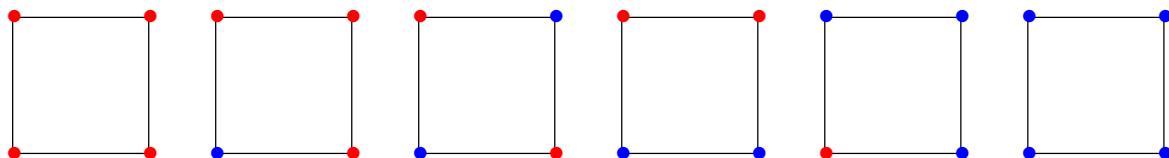
$$f_D = (1)(2, 4)(3)$$

$$f_A = (1, 3)(2)(4).$$

so the number of colorings is

$$f(k) = \frac{1}{8}(k^4 + k + k^2 + k + k^2 + k^2 + k^3 + k^3) = \frac{1}{8}(k^4 + 2k^3 + 3k^2 + 2k).$$

For example,  $f(2) = 6$ ; the six colorings are shown in figure 6.2.4. □



**Figure 6.2.4** The six 2-colorings of the square.

**EXAMPLE 6.2.7** Here is a more complicated example: how many different graphs are there on four vertices? In this case, of course, “different” means “non-isomorphic”. We can interpret this as a coloring problem: Color the *edges* of the complete graph  $K_4$  with

two colors, say black and white. The black edges form a graph; the white edges are the ones left out of the graph. The group  $G$  we need to consider is all permutations of the six edges of  $K_4$  induced by a permutation of the vertices, so  $|G| = 4! = 24$ . All we need to know is the number of cycles in each permutation; we consider a number of cases.

**Case 1.** The identity permutation on the vertices induces the identity permutation on the edges, with 6 cycles, so the contribution to the sum is  $2^6$ .

**Case 2.** A 4-cycle on the vertices induces a permutation of the edges consisting of one 4-cycle and one 2-cycle, that is, two cycles. There are  $3! = 6$  4-cycles on the vertices, so the contribution of all of these is  $6 \cdot 2^2$ .

**Case 3.** A permutation of the vertices consisting of a 3-cycle and a 1-cycle induces a permutation of the edges consisting of two 3-cycles. There are  $4 \cdot 2 = 8$  such permutations of the vertices, so the contribution of all is  $8 \cdot 2^2$ .

**Case 4.** A permutation of the vertices consisting of two 2-cycles induces a permutation of the edges consisting of two 1-cycles and two 2-cycles. There are  $\frac{1}{2} \binom{4}{2} = 3$  such permutations, so the contribution is  $3 \cdot 2^4$ .

**Case 5.** A permutation of the vertices consisting of a 2-cycle and two 1-cycles induces a permutation of the edges consisting of two 1-cycles and two 2-cycles. There are  $\binom{4}{2} = 6$  such permutations, so the contribution is  $6 \cdot 2^4$ .

The number of distinct colorings, that is, the number of distinct graphs on four vertices, is

$$\frac{1}{24}(2^6 + 6 \cdot 2^2 + 8 \cdot 2^2 + 3 \cdot 2^4 + 6 \cdot 2^4) = \frac{1}{24}(264) = 11.$$

□

It is possible, though a bit difficult, to see that for  $n$  vertices the result is

$$f(n) = \sum_{\mathbf{j}} \prod_{k=1}^n \frac{1}{k^{j_k} j_k!} \prod_{k=1}^{\lfloor n/2 \rfloor} 2^{kj_{2k}} \prod_{k=1}^{\lfloor (n-1)/2 \rfloor} 2^{kj_{2k+1}} \prod_{k=1}^{\lfloor n/2 \rfloor} 2^{kC(j_k, 2)} \prod_{1 \leq r < s \leq n-1} 2^{\gcd(r, s)j_r j_s} \quad (6.2.1)$$

where the sum is over all partitions  $\mathbf{j} = (j_1, j_2, \dots, j_n)$  of  $n$ , that is, over all  $\mathbf{j}$  such that  $j_1 + 2j_2 + 3j_3 + \dots + nj_n = n$ , and  $C(m, 2) = \binom{m}{2}$ . With this formula and a computer it is easy to compute the number of graphs when  $n$  is not too large; for example,  $f(5) = 34$ , so there are 34 different five-vertex graphs.

In light of the forgoing discussion, we can restate theorem 6.2.5. If  $\sigma$  is a permutation, let  $\#\sigma$  denote the number of cycles when  $\sigma$  is written in cycle form.

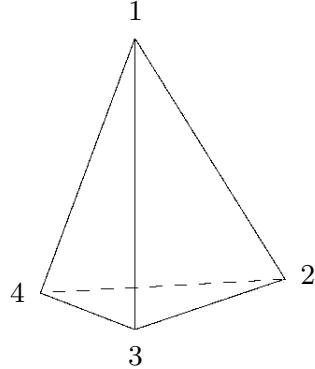
**COROLLARY 6.2.8** If group  $G$  acts on the colorings of an object, the number of distinct colorings modulo  $G$  with  $k$  colors is

$$\frac{1}{|G|} \sum_{\sigma \in G} k^{\#\sigma}.$$

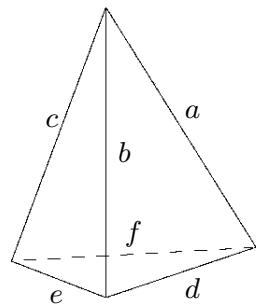
■

**Exercises 6.2.**

1. Write the 12 permutations of the vertices of the regular tetrahedron corresponding to the 12 rigid motions of the regular tetrahedron in cycle form. Use the labeling below.



2. Find the number of different colorings of the vertices of a regular tetrahedron with  $k$  colors, modulo the rigid motions.
3. Write the 12 permutations of the edges of the regular tetrahedron corresponding to the 12 rigid motions of the regular tetrahedron in cycle form. Use the labeling below.



4. Find the number of different colorings of the edges of a regular tetrahedron with  $k$  colors, modulo the rigid motions.
5. Find the number of non-isomorphic graphs on 5 vertices “by hand”, that is, using the method of example 6.2.7.

### 6.3 PÓLYA-REDFIELD COUNTING

Suppose we are interested in a more detailed inventory of the colorings of an object, namely, instead of the total number of colorings we seek the number of colorings with a given number of each color.

**EXAMPLE 6.3.1** How many distinct ways are there to color the vertices of a regular pentagon modulo  $D_5$  so that one vertex is red, two are blue, and two are green?

We can approach this as before, that is, the answer is

$$\frac{1}{|D_5|} \sum_{\sigma \in D_5} |\text{fix}(\sigma)|,$$

where  $\text{fix}(\sigma)$  now means the colorings with one red, two blues, and two greens that are fixed by  $\sigma$ . No longer can we use the simple expression of corollary 6.2.8.

The identity permutation fixes all colorings, so we need to know how many colorings of the pentagon use one red, two blues, and two greens. This is an easy counting problem: the number is  $\binom{5}{2}\binom{3}{2} = 30$ .

If  $\sigma$  is a non-trivial rotation,  $|\text{fix}(\sigma)| = 0$ , since the only colorings fixed by a rotation have all vertices the same color.

If  $\sigma$  is a reflection, the single vertex fixed by  $\sigma$  must be red, and then the remaining 2-cycles are colored blue and green in one of two ways, so  $|\text{fix}(\sigma)| = 2$ .

Thus, the number of distinct colorings is

$$\frac{1}{10}(30 + 0 + 0 + 0 + 0 + 2 + 2 + 2 + 2) = 4.$$

□

What we seek is a way to streamline this process, since in general the computations of  $|\text{fix}(\sigma)|$  can be tedious. We begin by recasting the formula of corollary 6.2.8.

**DEFINITION 6.3.2** The **type** of a permutation  $\sigma \in S_n$  is  $\tau(\sigma) = (\tau_1(\sigma), \tau_2(\sigma), \dots, \tau_n(\sigma))$ , where  $\tau_i(\sigma)$  is the number of  $i$ -cycles in the cycle form of  $\sigma$ . □

Note that  $\sum_{i=1}^n \tau_i(\sigma) = \#\sigma$ . Now instead of the simple

$$\frac{1}{|G|} \sum_{\sigma \in G} k^{\#\sigma}$$

let us write

$$\frac{1}{|G|} \sum_{\sigma \in G} x_1^{\tau_1(\sigma)} x_2^{\tau_2(\sigma)} \cdots x_n^{\tau_n(\sigma)}.$$

If we substitute  $x_i = k$  for every  $i$ , we get the original form of the sum, but the new version carries more information about each  $\sigma$ .

Suppose we want to know the number of colorings fixed by some  $\sigma$  that use  $i$  reds and  $j$  blues, where of course  $i + j = n$ . Using ideas familiar from generating functions, consider the following expression:

$$(r + b)^{\tau_1(\sigma)} (r^2 + b^2)^{\tau_2(\sigma)} \cdots (r^n + b^n)^{\tau_n(\sigma)}.$$

If we multiply out, we get a sum of terms of the form  $r^p b^q$ , each representing some particular way of coloring the vertices of cycles red and blue so that the total number of red vertices is  $p$  and the number of blue vertices is  $q$ , and moreover this coloring will be fixed by  $\sigma$ .

When we collect like terms, the coefficient of  $r^i b^j$  is the number of colorings fixed by  $\sigma$  that use  $i$  reds and  $j$  blues. This means that the coefficient of  $r^i b^j$  in

$$\sum_{\sigma \in G} (r+b)^{\tau_1(\sigma)} (r^2+b^2)^{\tau_2(\sigma)} \cdots (r^n+b^n)^{\tau_n(\sigma)}$$

is

$$\sum_{\sigma \in G} |\text{fix}(\sigma)|$$

where  $\text{fix}(\sigma)$  is the set of colorings using  $i$  reds and  $j$  blues that are fixed by  $\sigma$ . Finally, then, the number of distinct colorings using  $i$  reds and  $j$  blues is this coefficient divided by  $|G|$ . This means that by multiplying out

$$\frac{1}{|G|} \sum_{\sigma \in G} (r+b)^{\tau_1(\sigma)} (r^2+b^2)^{\tau_2(\sigma)} \cdots (r^n+b^n)^{\tau_n(\sigma)}$$

and collecting like terms, we get a list of the number of distinct colorings using any combination of reds and blues, each the coefficient of a different term; we call this the **inventory** of colorings. If we substitute  $r = 1$  and  $b = 1$ , we get the sum of the coefficients, namely, the total number of distinct colorings with two colors.

**DEFINITION 6.3.3** The **cycle index** of  $G$  is

$$\mathcal{P}_G = \frac{1}{|G|} \sum_{\sigma \in G} \prod_{i=1}^n x_i^{\tau_i(\sigma)}.$$

□

**EXAMPLE 6.3.4** Consider again example 6.2.6, in which we found the number of colorings of a square with two colors. The cycle index of  $D_4$  is

$$\frac{1}{8}(x_1^4 + x_4^1 + x_2^2 + x_4^1 + x_2^1 + x_2^2 + x_1^2 x_2 + x_1^2 x_2) = \frac{1}{8}x_1^4 + \frac{1}{4}x_1^2 x_2 + \frac{3}{8}x_2^2 + \frac{1}{4}x_4.$$

Substituting as above gives

$$\frac{1}{8}(r+b)^4 + \frac{1}{4}(r+b)^2(r^2+b^2) + \frac{3}{8}(r^2+b^2)^2 + \frac{1}{4}(r^4+b^4) = r^4 + r^3b + 2r^2b^2 + rb^3 + b^4.$$

Thus there is one all red coloring, one with three reds and one blue, and so on, as shown in figure 6.2.4. □

There is nothing special about the use of two colors. If we want to use three colors, we substitute  $r^i + b^i + g^i$  for  $x_i$  in the cycle index, and for  $k$  colors we substitute something like  $c_1^i + c_2^i + c_3^i + \cdots + c_k^i$ .

**EXAMPLE 6.3.5** Let's do the number of 3-colorings of the square. Since we already have the cycle index, we need only substitute  $x_i = r^i + b^i + g^i$  and expand. We get

$$\begin{aligned} \frac{1}{8}(r+b+g)^4 + \frac{1}{4}(r+b+g)^2(r^2+b^2+g^2) + \frac{3}{8}(r^2+b^2+g^2)^2 + \frac{1}{4}(r^4+b^4+g^4) \\ = b^4 + b^3g + b^3r + 2b^2g^2 + 2b^2gr + 2b^2r^2 + bg^3 + 2bg^2r + 2bgr^2 + \\ br^3 + g^4 + g^3r + 2g^2r^2 + gr^3 + r^4. \end{aligned}$$

So, for example, there are two squares with two blue vertices, one green, and one red, from the  $b^2gr$  term.  $\square$

**EXAMPLE 6.3.6** Consider again example 6.2.7, in which we counted the number of four-vertex graphs. Following that example, we get

$$\mathcal{P}_G = \frac{1}{24}(x_1^6 + 6x_2x_4 + 8x_3^2 + 3x_1^2x_2^2 + 6x_1^2x_2^2),$$

and substituting for the variables  $x_i$  gives

$$r^6 + r^5b + 2r^4b^2 + 3r^3b^3 + 2r^2b^4 + rb^5 + b^6.$$

Recall that the “colors” of the edges in this example are “included” and “excluded”. If we set  $b = 1$  and  $r = i$  (for “included”) we get

$$i^6 + i^5 + 2i^4 + 3i^3 + 2i^2 + i + 1,$$

interpreted as one graph with 6 edges, one with 5, two with 4, three with 3, two with 2, one with 1, and one with zero edges, since  $1 = i^0$ .  $\square$

It is possible, though a bit difficult, to see that for  $n$  vertices the cycle index is

$$\mathcal{P}_G = \sum_{\mathbf{j}} \prod_{k=1}^n \frac{1}{k^{j_k} j_k!} \prod_{k=1}^{\lfloor n/2 \rfloor} (x_k x_{2k}^{k-1})^{j_{2k}} \prod_{k=1}^{\lfloor (n-1)/2 \rfloor} x_{2k+1}^{kj_{2k+1}} \prod_{k=1}^{\lfloor n/2 \rfloor} x_k^{kC(j_k, 2)} \prod_{1 \leq r < s \leq n-1} x_{\text{lcm}(r,s)}^{\gcd(r,s)j_r j_s},$$

where the sums are over all partitions  $\mathbf{j} = (j_1, j_2, \dots, j_n)$  of  $n$ , that is, over all  $\mathbf{j}$  such that  $j_1 + 2j_2 + 3j_3 + \dots + nj_n = n$ , and  $C(m, 2) = \binom{m}{2}$ . This is where the formula 6.2.1 comes from, substituting  $x_i = 2$  for all  $i$ .

With this formula and a computer it is easy to compute the inventory of  $n$ -vertex graphs when  $n$  is not too large. When  $n = 5$ , the inventory is

$$i^{10} + i^9 + 2i^8 + 4i^7 + 6i^6 + 6i^5 + 6i^4 + 4i^3 + 2i^2 + i + 1.$$

***Exercises 6.3.***

1. Find the cycle index  $\mathcal{P}_G$  for the group of permutations of the vertices of a regular tetrahedron induced by the rigid motions. (See exercise 1 in section 6.2.)
2. Using the previous exercise, write out a full inventory of colorings of the vertices of a regular tetrahedron induced by the rigid motions, , with three colors, as in example 6.3.5. You may use Sage or some other computer algebra system.
3. Find the cycle index  $\mathcal{P}_G$  for the group of permutations of the edges of  $K_5$ . (See exercise 5 in section 6.2. Don't use the general formula above.)
4. Using the previous exercise, write out a full inventory of the graphs on five vertices, as in example 6.3.6. You may use Sage or some other computer algebra system.

# A

## Hints

- 1.6.2.** Every positive integer can be written as  $j \cdot 2^k$ , with  $j$  odd and  $k \geq 0$ .



# Index

## A

action  
group action on a set, 138  
acyclic graph, 105  
addition principle, 12  
adjacent, 83  
alternating chain, 85  
anti-chain, 36  
arc, 129

## B

Bell numbers, 22  
Bell triangle, 25  
binomial coefficients, 16  
monotonicity, 19  
symmetry, 19  
bipartite graph, 84, 103  
complete, 11, 104, 117  
block, 116  
block-cutpoint graph, 117  
bridge, 106  
Brooks's Theorem, 121

## C

Catalan numbers, 16, 66  
chain, 36  
Chinese Remainder Theorem, 33  
chromatic number, 118  
chromatic polynomial, 125  
circuit, 98, 129

class, 120  
clique, 95, 119  
clique number, 119  
closed neighborhood, 83  
closed walk, 98, 129  
complete bipartite graph, 11, 104, 117  
complete graph, 10, 34, 95  
 $K_n$ , 84  
condensation, 91  
conjugate of a partition, 61  
connected, 91  
connected components, 95  
cut, 110  
in network, 131  
cutpoint, 107, 110  
cutset, 110  
cycle, 36, 95  
cycle form, 143  
cycle form of a permutation, 39  
cycle graph ( $C_n$ ), 83  
cycle index, 148  
cyclic group, 138

## D

degree, 83  
maximum, 120  
minimum, 111  
derangement, 48  
derangement numbers, 49  
digraph, 129  
connected, 134  
strongly connected, 134

underlying graph, 134  
dihedral group, 138  
directed edge (arc), 129  
directed graph (digraph), 129

***E***

endblock, 117  
equivalence relation, 95  
Euler circuit, 98  
Euler walk, 99  
exponential generating function, 57

***F***

falling factorial, 43  
Ferrers diagram, 61  
flow, 129  
value of, 131  
forest, 105

***G***

Galton board, 21  
general graph, 91  
generating function, 53  
exponential, 57  
girth, 119  
graph  
directed, 129  
weighted, 108  
graphical parameters, 119  
graphical sequence, 93  
greedy algorithm, 120  
group, 138  
cyclic, 138  
dihedral, 138  
symmetric, 138

***H***

Hall's Condition, 72  
Hall's Theorem, 72  
Hamilton cycle, 100  
Hamilton path, 100  
in digraph, 134  
handle, 115  
Handle Theorem, The, 115

***I***

incident, 9, 83

indegree, 129  
independence number, 118  
independent set, 118  
induced subgraph, 95  
internally disjoint, 113  
inventory, 148  
isomorphic, 94  
isomorphism, 94  
isotopic, 78  
isotopy class, 78

***J***

Jarník Algorithm, 108

***K***

Kronecker delta, 41  
Kruskal's Algorithm, 109

***L***

Latin square, 76  
isotopic, 78  
isotopy class, 78  
orthogonal, 79  
reduced, 77  
least cost spanning tree, 108  
length, 83  
loop, 83, 91

***M***

matching, 84  
perfect, 105  
maximum degree, 120  
maximum flow, 131  
minimum degree, 111  
modulo  
colorings modulo  $G$ , 139  
multigraph, 83, 91  
multinomial coefficient, 29  
multiple edges, 91  
multiplication principle, 12  
multiset, 28  
multitree, 108

***N***

neighborhood, 83  
closed, 83  
open, 83  
network, 129

*n*-set, 14

## O

open neighborhood, 83  
orbit, 139  
Ore property, 102  
outdegree, 129

## P

partition  
conjugate, 61  
non-crossing, 69  
of a set, 22, 69  
of an integer, 59  
self-conjugate, 62  
Pascal's Triangle, 14  
monotonicity, 19  
symmetry, 19  
path, 83, 95  
in digraph, 129  
pendant, 105  
pendant vertex, 94  
perfect matching, 105  
permutation, 36  
cycle form, 39  
permutations, 13  
Petersen graph, 103  
pigeonhole principle, 32  
planar, 125  
Prim's Algorithm, 108  
proper coloring, 118

## Q

quasigroup, 78

## R

Ramsey number, 34  
Ramsey Theory, 35  
recurrence relation, 22, 50, 62  
regular graph, 105  
repetition number, 28  
rising factorial, 43  
rooted binary tree, 66

## S

SDR, 71  
separating set, 113  
sequence, 92

set system, 71  
simple graph, 91  
source, 129  
spanning tree, 106  
least cost, 108  
Stirling numbers of the first kind, 39, 40  
unsigned, 40  
Stirling numbers of the second kind, 27, 39  
subgraph, 95  
symmetric group, 138  
system of distinct representatives, 71

## T

target, 129  
tree, 105  
rooted binary, 66  
type, 147

## V

value of a flow, 131

## W

walk, 98  
in digraph, 129  
weighted graph, 108