Chapter 4

Graph Theory

We began this course with a graph theoretic model for solving a committee assignment problem. A large amount of modern discrete mathematics relies on graph theoretic constructions, and many modeling problems naturally lend themselves to a graph representation. We will spend some time understanding graph structures, their properties, and how to solve problems using them.

4.1 Definitions

A graph is a structure with vertices (nodes) and edges (links) connecting them. But there are many flavors of graphs. Some have undirected edges and some have directed edges. Some are restricted to single edges between vertices and some allow multiple edges between vertices. On some special graphs, such as cycle graphs and bipartite graphs, we can solve problems more easily, and so the challenge can sometimes be identifying these special cases. We begin our study of graph theory with definitions of the basic structures.

Definition 4.1 (Simple Graph)**.** *A simple graph $G = (V, E)$ is a set V of vertices, and a set E of edges that are unordered pairs of distinct elements of V:*

$$\text{If } e \in E,\ e = \{u, v\},\ \text{where } u, v \in V, u \neq v.$$

Example 4.2. A simple graph:

Definition 4.3 (Multigraph)**.** *A multigraph $G = (V, E)$ is a set V of vertices, a set E of edges, and a function*

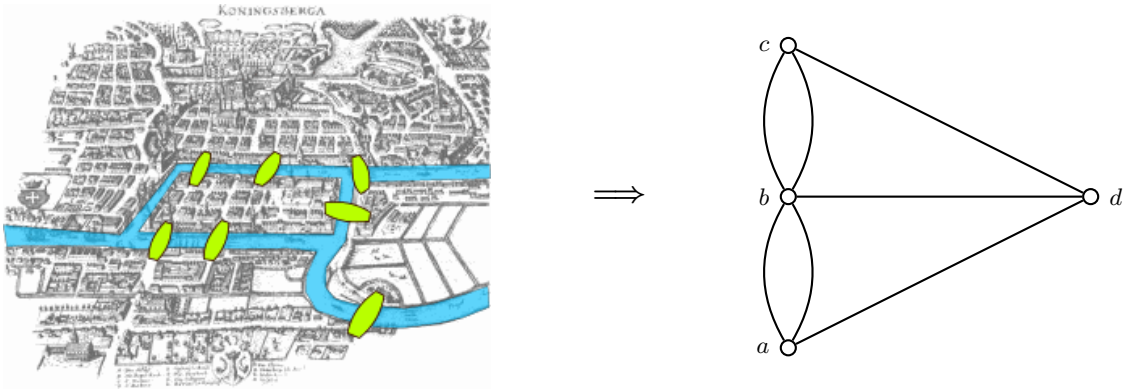
$$f : E \to \{\{u, v\} : u, v \in V, u \neq v\}.$$

Note that in a multigraph, unlike for a simple graph, more than one edge can be mapped to two given vertices u and v. That is why the function f is needed in the definition.

Definition 4.4 (Multiple edges)**.** *Two edges e_1 and e_2 are called* multiple *or* parallel *edges if the function f assigns both of them to the same unordered pair of vertices $\{u, v\}$:*

$$f(e_1) = f(e_2).$$

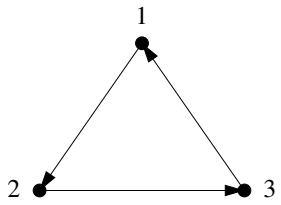
Example 4.5 (The seven bridges of Königsberg)**.** This famous problem, solved by Euler in 1736, asks whether seven bridges connecting four different land masses can all be traveled in a single trip, without crossing any bridge more than once, and with the trip ending where it began. The problem is solved by modeling the bridges as a multigraph:



Definition 4.6 (Directed graph)**.** *A directed graph (digraph) $D = (V, E)$ is a set V of vertices V, and a set E of edges that are* ordered *pairs of distinct elements of V:*

$$\textit{If } e \in E,\ e = (u, v),\ \textit{where } u, v \in V, u \neq v.$$

Example 4.7. The periodic 3-state Markov chain that we saw earlier is an example of a directed graph:



Here are some further examples of the use of such graph structures in discrete modeling.

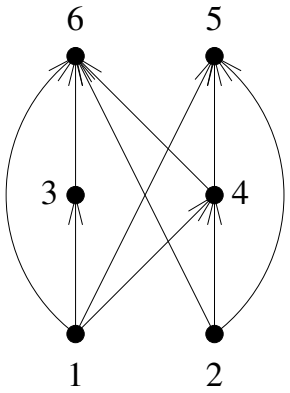
- Electrical circuits: Kirchhoff's laws relating currents and voltages in circuits were an early and very powerful application of graph theory.
- Communication networks.
- Social networks: collaboration graphs, gmail contact graphs, etc.
- Web graph.
- Ecosystem: directed graph of who eats whom.

A final example involves the use of a directed graph to determine which computational operations can be performed concurrently.

Example 4.8 (Precedence graphs: Rosen 9.1, example 9)**.** Consider a program that performs the following variable assignments:

1. $a := 0$
2. $b := 1$
3. $c := a + 1$
4. $d := a + b$
5. $e := d + 1$
6. $f := c + d$

Can certain of these operations be performed concurrently in a multi-processor machine? To answer the question, we construct a directed graph where each of the assignments above is a vertex, and a directed edge (u, v) means that assignment u must be performed *before* assignment v. The resulting graph looks like



So, for instance, assignment 5 cannot be executed until after assignments 1, 2 and 4. On the other hand, assignments 5 and 3 could be performed simultaneously by different processors, and the same holds for assignments 5 and 6.

4.2 Properties

We now turn to some basic properties on graphs.

4.2.1 Adjacency and degree

Definition 4.9 (Adjacency)**.** *Given an undirected graph $G = (V, E)$, vertices $u, v \in V$ are called* adjacent *or* neighbors *if there is an edge between them:*

$$\exists e \in E : e = \{u, v\}$$

in a simple graph, or

$$\exists e \in E : f(e) = \{u, v\}$$

in a multigraph.

Definition 4.10 (Incidence and endpoint)**.** *Given an undirected graph $G = (V, E)$, edge $e \in E$ is* incident *on vertices $u, v \in V$ if $e = \{u, v\}$ (or $f(e) = \{u, v\}$ in a multigraph). Vertices u and v are* endpoints *of e.*

Definition 4.11 (Degree)**.** *Given a simple graph $G = (V, E)$, the* degree *of a vertex $v \in V$ is the number of edges incident on it:*

$$\deg(v) = \big|\{u : \{u, v\} \in E\}\big|.$$

Note that for a multigraph, the notation is a bit more complex since we must count not simply the number of neighbors of v but rather the number of edges with one endpoint at v:

$$\deg(v) = \big|\{e : \exists u : f(e) = \{u, v\}\}\big|.$$

From these definitions, we see a fundamental relation between degree and the edge set of the graph.

Theorem 4.12 (Handshaking theorem)**.** *Given an undirected graph $G = (V, E)$,*

$$\sum_{v \in V} \deg(v) = 2|E|.$$

The factor of two on the right-hand side holds because we are double-counting edges: we count an edge once at each of its endpoints.

Example 4.13. Consider the multigraph of the bridges of Königsberg. The degrees of the nodes are 3, 5, 3 and 3, so the sum of all degrees is 14. There are 7 edges in the graph, so $2|E| = 14$.

Corollary 4.14. *An undirected graph $G = (V, E)$ has an even number of vertices with odd degree.*

For directed graphs, there are some analogous though slightly more complicated definitions.

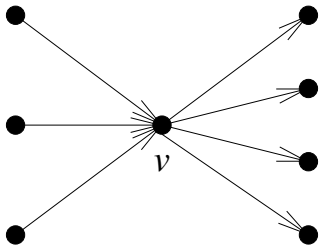
Definition 4.15 (In-degree)**.** *Given a directed graph $D = (V, E)$, the* in-degree *of a vertex $v \in V$ is the number of directed edges terminating at it:*

$$\deg^-(v) = \big|\{u : (u, v) \in E\}\big|.$$

Definition 4.16 (Out-degree)**.** *Given a directed graph $D = (V, E)$, the* out-degree *of a vertex $v \in V$ is the number of directed edges originating at it:*

$$\deg^+(v) = \big|\{u : (v, u) \in E\}\big|.$$

Example 4.17. In the graph below, node v has in-degree $\deg^-(v) = 3$ and out-degree $\deg^+(v) = 4$.



The handshaking theorem also generalizes to directed graphs.

Theorem 4.18 (Handshaking theorem for directed graphs)**.** *Given a directed graph $D = (V, E)$,*

$$\sum_{v \in V} \deg^+(v) = \sum_{v \in V} \deg^-(v) = |E|.$$

4.2.2 Subgraphs

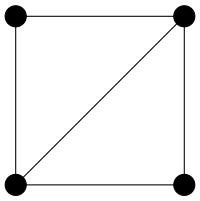
Often, we are interested in characterizing some part of a graph. In order to do so, we introduce the notion of a subgraph.

Definition 4.19 (Subgraph)**.** *A subgraph of $G = (V, E)$ is a graph $H = (W, F)$ where $W \subseteq V$ and $F \subseteq E$.*

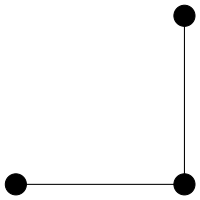
Definition 4.20 (Induced subgraph)**.** *A subgraph of $G = (V, E)$* induced by *$W \subseteq V$ is $H = (W, F)$, where F is the set of all edges $e \in E$ that have both endpoints in W:*

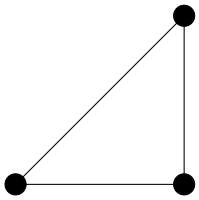
$$F = \{\{u, v\} \in E : u, v \in W\}.$$

Example 4.21. Consider a graph G:



If the vertex set W consists of all but the upper left vertex, then a possible subgraph with vertices W is shown below on the left, whereas the *induced* subgraph on the vertices W is shown below on the right.





4.2.3 Paths and connectivity

A crucial question on graphs is whether or not it is possible to reach one vertex from another. For instance, say a node in a communication network tries to send a message to another node. Can it do so? Or given a set of WWW links, is it possible to get from one website to another simply by clicking on a series of links?

To start with, we define the notion of a *path*.

Definition 4.22 (Path)**.** *Given an undirected graph $G = (V, E)$, a* path *of length n between $u \in V$ and $v \in V$ is a sequence of edges $e_1, e_2, \ldots, e_n \in E$ such that*

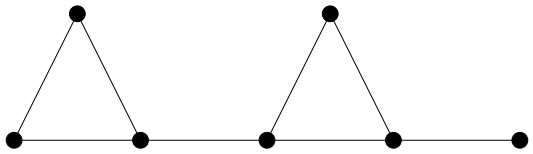
$$e_1 = \{u, x_1\}, e_2 = \{x_1, x_2\}, \ldots, e_n = \{x_{n-1}, v\},$$

for some $x_1, \ldots, x_{n-1} \in V$.

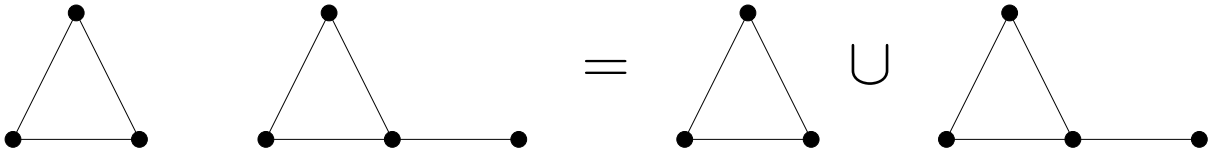
Equipped with this, we can define whether a graph is *connected*.

Definition 4.23 (Connectivity)**.** *An undirected graph $G = (V, E)$ is* connected *if $\forall u \neq v \in V$, there is a path between u and v.*

If a graph is connected, it is all one single cluster, or *component*. An example is the following:



Conversely, if a graph is not connected, it is necessarily the union of multiple components:



The notion of a path is generalized to directed graphs in an entirely straightforward manner.

Definition 4.24 (Directed path)**.** *Given a directed graph $D = (V, E)$, a* directed path *of length n from $u \in V$ to $v \in V$ is a sequence of edges $e_1, e_2, \ldots, e_n \in E$ such that*

$$e_1 = (u, x_1),\ e_2 = (x_1, x_2),\ \ldots,\ e_n = (x_{n-1}, v),$$

for some $x_1, \ldots, x_{n-1} \in V$.

The notion of connectivity is a little more subtle, though.

Definition 4.25 (Strong connectivity)**.** *A directed graph $D = (V, E)$ is* strongly connected *if $\forall u, v \in V$, there is a directed path from u to v.*

Example 4.26 (Markov chain)**.** Given a Markov chain with transition matrix P, define its underlying directed graph D as follows. Represent every state by a vertex, and place a directed edge from u to v if $P_{uv} > 0$. Then, D is strongly connected if and only if P is irreducible.

Weak connectivity holds if we cannot guarantee that there is a directed path between any two vertices, but merely an undirected path:

Definition 4.27 (Weak connectivity)**.** *Given a directed graph $D = (V, E)$, consider the corresponding undirected graph $G = (V, F)$: if $(u, v) \in E$, where $u, v \in V$, let $\{u, v\} \in F$. Then the directed graph D is* weakly connected *if G is connected.*

4.2.4 Special cases

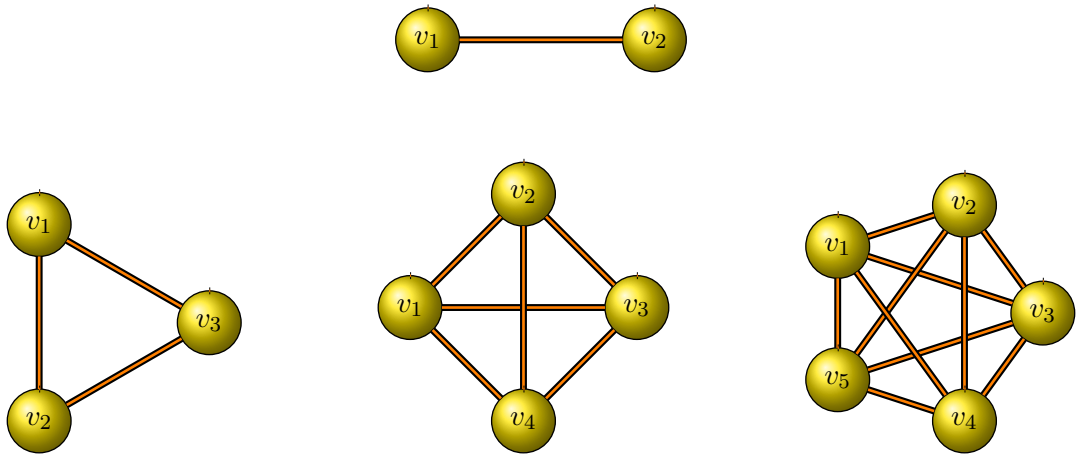
Finally, we consider two special types of graphs.

Definition 4.28 (Complete graph)**.** *The* complete graph *or* clique *$K_n = (V, E)$ is the simple graph on $|V| = n$ vertices where E consists of all unordered pairs of distinct elements of V:*

$$\forall u \neq v \in V, \quad \{u, v\} \in E.$$

Thus, in a complete graph, each node is adjacent to all other nodes.

Example 4.29. The complete graphs K_2 through K_5 are

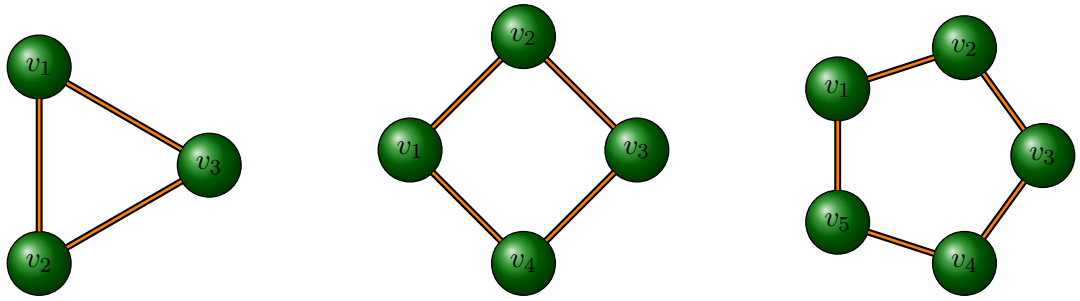


The other special type of graph that we define is the cycle graph:

Definition 4.30 (Cycle graph)**.** *The* cycle graph $C_n = (V, E)$ *is the simple graph on* $|V| = n \geq 3$ *vertices* $v_1, \ldots, v_n$*, and* n *edges*

$$E = \big\{\{v_1, v_2\}, \{v_2, v_3\}, \ldots, \{v_{n-1}, v_n\}, \{v_n, v_1\}\big\}.$$

Example 4.31. The cycle graphs C_3 through C_5 are



4.3 Graph coloring

We now turn to a classic graph theoretic problem: graph coloring. Recall the committee scheduling example that we saw at the very beginning of this course. We modeled it as a graph coloring problem, where vertices are assigned colors such that no two adjacent vertices have the same color. Let us provide some formal definitions for the problem, and consider it on a few specific kinds of graphs. Later on, we will look at more general algorithms for solving it.

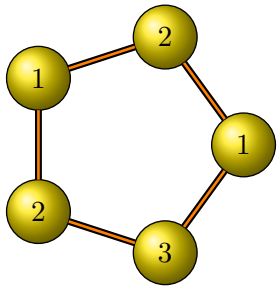
4.3.1 Definitions

Definition 4.32 (Coloring)**.** *A* coloring *of a simple graph* $G = (V, E)$ *assigns to each vertex* $v \in V$ *a value, or* color*,* $c(v)$*, such that no two adjacent vertices are assigned the same color:*

$$\forall \{u, v\} \in E, \quad c(u) \neq c(v).$$

Definition 4.33 (Chromatic number)**.** *The* chromatic number $\chi(G)$ *of a simple graph* $G = (V, E)$ *is the minimum number of colors needed for a coloring of the graph.*

Example 4.34. Consider assigning colors to the cycle graph C_5:



We can do this with 3 colors, but there is no way to use fewer colors without a conflict, so $\chi(C_5) = 3$.

In general, finding the chromatic number of a graph is not easy. But for a few special kinds of graphs, it is straightforward.

4.3.2 Cliques

In a complete graph, any two vertices are adjacent to each other. This has the following consequence on graph coloring.

Theorem 4.35. *The chromatic number of the complete graph K_n is $\chi(K_n) = n$.*

Proof. It is always possible to color n vertices with n distinct colors, so $\chi(K_n) \leq n$. Assume that $\chi(K_n) < n$. If $n-1$ colors or fewer are used to color n vertices, then by the pigeonhole principle, at least two vertices must have the same color. But since K_n is a complete graph, those two vertices must be adjacent, so this cannot be a valid coloring. Hence, $\chi(K_n) = n$. □

It therefore follows that the presence of a clique within any graph provides a bound on the chromatic number.

Corollary 4.36. *If a simple graph G contains a clique of size n as a subgraph ($K_n \subseteq G$), then $\chi(G) \geq n$.*

This motivates the definition of the *clique number*:

Definition 4.37 (Clique number)**.** *The* clique number $\omega(G)$ *of a simple graph G is the size of the largest clique found within the graph:*

$$\omega(G) = \max\{n : K_n \subseteq G\}.$$

And finally, it follows that the clique number bounds the chromatic number.

Corollary 4.38. *For any simple graph G,*

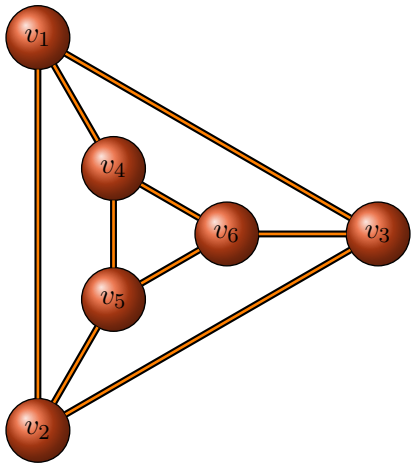
$$\chi(G) \geq \omega(G).$$

4.3.3 Planar graphs

A very useful class of graphs, frequently encountered among networks in a geometric setting, is defined as follows.

Definition 4.39 (Planar graph)**.** *A graph is planar if it can be represented in the plane with no two edges crossing.*

Example 4.40. Consider the following graph with $|V| = 6$ and $|E| = 9$:



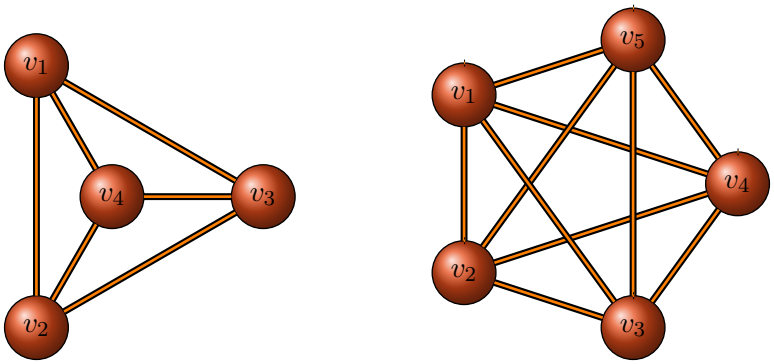
Since no two edges cross each other, the graph is planar.

A powerful theorem bounding the chromatic number of planar graphs was proved by Appel and Haken in 1976, after remaining an open conjecture for over a century.

Theorem 4.41 (Four color theorem)**.** *Every planar graph G can be colored with four colors or less:*

$$\chi(G) \leq 4.$$

Example 4.42. Consider K_4, which is planar, and K_5, which is not:



Since $\chi(K_n) = n$, planar K_4 has a chromatic number of 4, whereas nonplanar K_5 has a chromatic number greater than 4.

Example 4.43 (Cycle graphs)**.** It is easy to see that every cycle graph C_n is planar, so

$$\forall n, \quad \chi(C_n) \leq 4.$$

4.3.4 Bipartite graphs

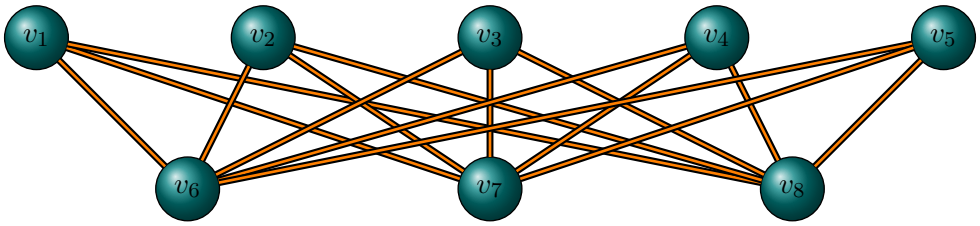
Another frequently encountered class of graphs is the following.

Definition 4.44 (Bipartite graph)**.** *A simple graph $G = (V, E)$ is* bipartite *if V can be partitioned into two disjoint subsets V_1, V_2, such that every edge in the graph connects a vertex in V_1 with a vertex in V_2:*

$$\exists V_1, V_2 : \ V = V_1 \cup V_2, \ V_1 \cap V_2 = \emptyset, \ \text{and } \forall \{v_1, v_2\} \in E, \ \text{either } v_1 \in V_1 \text{ and } v_2 \in V_2 \text{ or vice-versa.}$$

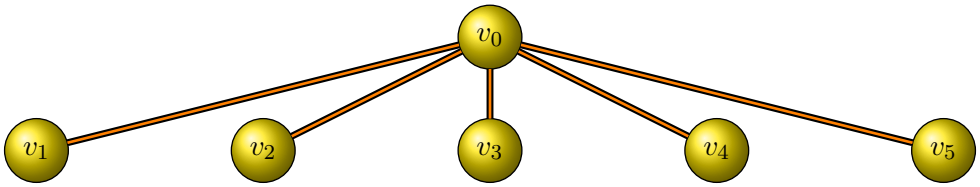
Thus, in a bipartite graph, no edge connects two vertices in the same subset.

Example 4.45. Consider the following graph, with $V_1 = \{v_1, v_2, v_3, v_4, v_5\}$ and $V_2 = \{v_6, v_7, v_8\}$:



No edge connects any two vertices within V_1 or any two vertices within V_2, so the graph is bipartite.

Example 4.46 (Star graph)**.** A special case of a bipartite graph is the star graph, S_n, in which $|V_1| = 1$ and $|V_2| = n$. For instance, if $n = 5$:



Bipartite graphs are particularly simple to color:

Theorem 4.47. *A simple graph G is bipartite iff $\chi(G) = 2$.*

The proof is simply a matter of equating color 1 with all vertices in V_1 and color 2 with all vertices in V_2.

Example 4.48 (Cycle graphs, revisited)**.** We noted earlier that for any cycle graph C_n, planarity implies that $\chi(C_n) \leq 4$. But C_n is bipartite as long as n is even, since we can assign all odd-numbered vertices to V_1 and all even-numbered vertices to V_2. So we can improve upon the chromatic number bound:

$$\chi(C_n) = 2 \quad \text{for } n \text{ even.}$$

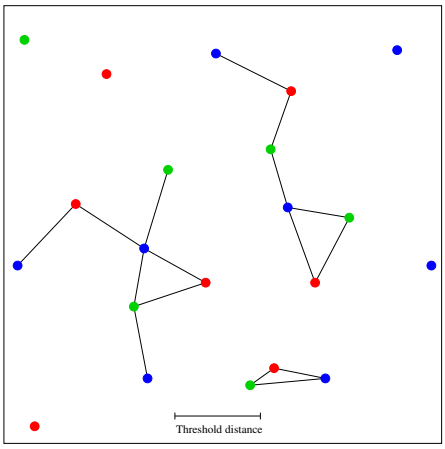
It is not hard to see that if the cycle is of odd length, one additional color is sufficient, so

$$\chi(C_n) = 3 \quad \text{for } n \text{ odd.}$$

4.3.5 Applications

Here are some examples of the graph coloring problem in discrete modeling applications:

- Scheduling problems: for instance, the committee assignment problem that we have been referring to throughout. **The chromatic number is the minimum number of possible meeting times.**
- Channel assignment: in radio communication, no two transmitters can broadcast on the same channel if they are too close, or else interference will result. What is the minimum number of channels needed? The figure below shows locations of transmitters. An edge is placed between two nodes if the transmitters are within a threshold distance of each other and therefore cannot use the same channel. A possible assignment of channels is shown, with node colors representing channels.



The chromatic number is the minimum number of possible channels.

- Cache memory management: computers will typically put a variable value into cache memory if that variable needs to be accessed frequently. How much cache is needed? Represent each variable by a vertex in a graph, and place an edge between two vertices if the two variables could need to be accessed at the same time. **The chromatic number is the number of spaces needed in cache.**

4.4 Trees

Trees are an important subclass of bipartite graphs. A tree is a graph that contains no cycles: it can only have a branching structure.

4.4.1 Definitions

Definition 4.49 (Tree)**.** *A simple, connected graph G is a* tree *if it does not have any cycles as subgraphs:*

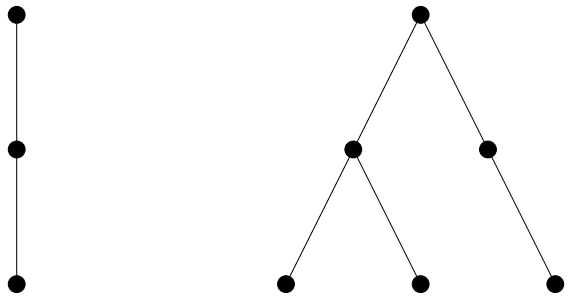
$$\forall n \geq 3, \quad C_n \nsubseteq G.$$

If the graph is not connected, it is the union of connected components. If each of these components is a tree, the graph is known as a *forest*.

Definition 4.50 (Forest)**.** *A simple graph G is a* forest *if it does not have any cycles as subgraphs:*

$$\forall n \geq 3, \quad C_n \nsubseteq G.$$

Example 4.51. The graph on the left below and the graph on the right below are both trees. Together, they form a forest.



Example 4.52 (Saturated hydrocarbons)**.** The graph theoretic notion of trees dates back to 1857, when the English mathematician Arthur Cayley used them for counting different types of chemical compounds. A hydrocarbon is a molecule containing carbon and hydrogen atoms: C_nH_{2n+2}. Carbon atoms bond to four neighbors while hydrogen atoms bond to only one neighbor. Graph theoretically, carbon may be modeled as a vertex of degree 4, and hydrogen as a vertex of degree 1. Counting up carbon and hydrogen atoms, the total number of vertices is

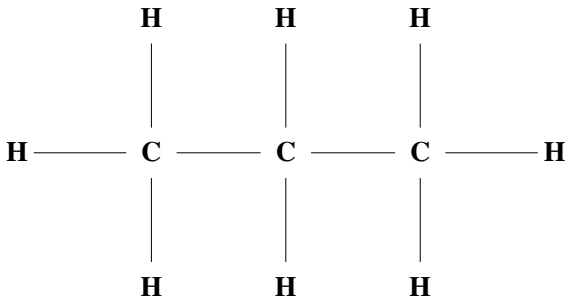
$$|V| = n + (2n + 2) = 3n + 2.$$

Counting up the degrees of all vertices, by the handshaking theorem, the total number of edges is

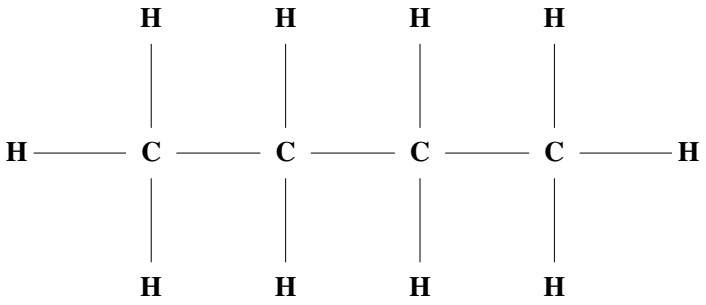
$$|E| = \frac{1}{2}[4n + (2n + 2)] = 3n + 1$$

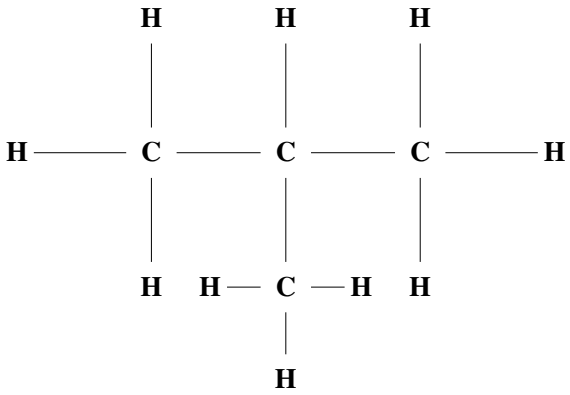
As we will see later, for this to be the case, the graph must be a tree.

Cayley showed that if $n = 3$, there is only one possible molecular configuration, known as *propane*:



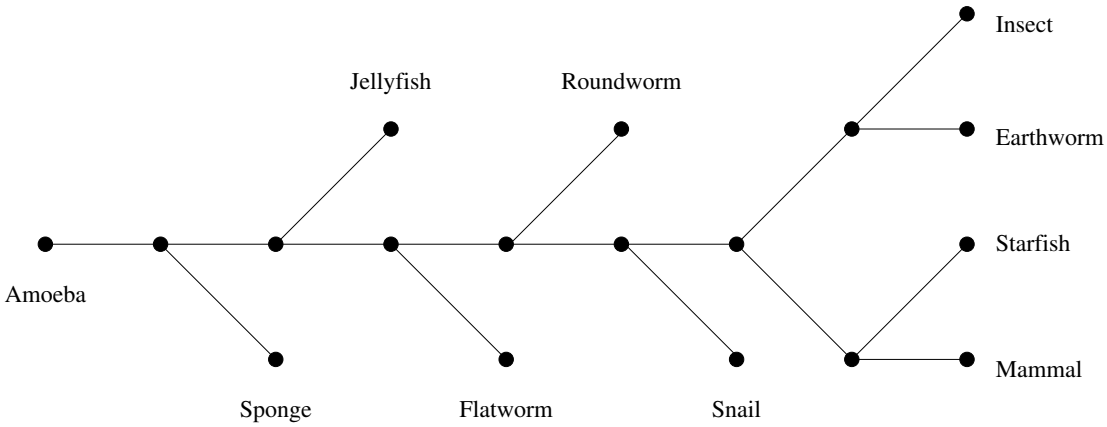
If $n = 4$, two molecular configurations are possible (*isomers*), known as *butane* and *isobutane*:





Since Cayley's time, trees have been employed to solve problems in a wide variety of disciplines. Trees are particularly useful in computer science where, for instance, they are used to construct efficient algorithms for locating items in a list. A related application is in biology, for modeling the evolution of species.

Example 4.53 (Phylogenetic trees)**.** Let species be vertices in a graph, with edges connecting ancestors to descendants.



Often, the goal is to reconstruct the most probable tree based on data describing the nodes, such as genetic information. From this, one can determine what is, for instance, the common ancestor of two species. The technique of reconstructing phylogenetic trees is useful in other areas as well, such as in linguistics (finding common ancestor of two languages) where it can provide evidence of population migration.

4.4.2 Properties

We now consider a number of fundamental graph theoretic properties of trees.

Theorem 4.54. *Given a tree $T = (V, E)$,*

$$|E| = |V| - 1.$$

Proof. Clearly, when $|V| = 1$ and $|V| = 2$, the theorem holds. The rest of the proof is by induction.

Take the tree T and add a new vertex u to create a new tree $\hat{T}$. Clearly, since trees are connected, we must add at least one new edge with endpoint u. Suppose that we use more than one edge. In that case, $\hat{T}$ has edge $\{u, v\}$ and edge $\{u, w\}$, for some $v \neq w \in V$. But since T is connected, it must already have a path connecting v to w, so $\hat{T}$ has a cycle. Thus, by contradiction, it follows that adding one vertex to the tree must add exactly one edge. □

Later on, we will prove the converse of this theorem.

A few further useful properties of trees follow.

Corollary 4.55. *Given a tree $T = (V, E)$ with $|V| > 1$, at least two vertices have degree 1.*

Proof. Assume the contrary. If at most one vertex has degree 1, then all the others have degree 2 or more, so

$$\sum_{v \in V} \deg(v) \geq 2|V| - 1.$$

But we know from the handshaking theorem that

$$\sum_{v \in V} \deg(v) = 2|E|,$$

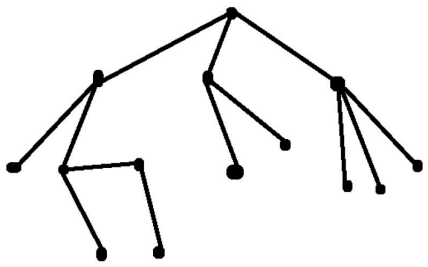
and $|E| = |V| - 1$ since the graph is a tree, so

$$\sum_{v \in V} \deg(v) = 2|V| - 2.$$

which contradicts our first inequality. Thus, at least two vertices must have degree 1. □

Theorem 4.56. *Every tree is planar.*

Proof. Let any vertex be the "root" of the tree. Place its neighbors ("descendants") from left to right below it, and repeat this operation for the neighbors of those vertices. Iterate until all vertices have been used. The result will be as shown below, with no edges crossing:



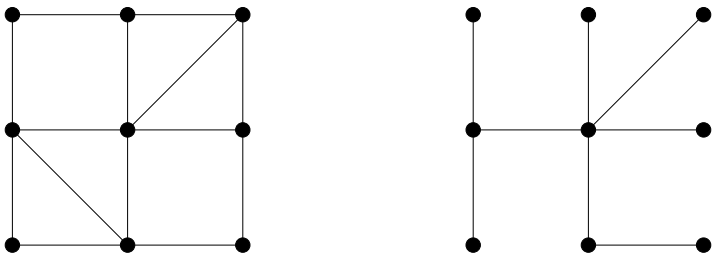
□

4.4.3 Spanning trees

Given a graph, we may want to find a subgraph of it that is a tree.

Definition 4.57 (Spanning tree)**.** *Given a simple graph $G = (V, E)$, a spanning tree $T \subseteq G$ is a tree with vertex set V.*

Example 4.58. Imagine a road network connecting a number of cities. During a major snowstorm, we would like to find a limited number of roads to plow so that there is nevertheless an accessible path from any one city to another. The figure on the left shows a road network modeled as a simple graph. The figure on the right shows a spanning tree of the graph. Clearly, the spanning tree gives a sufficient collection of roads to plow.



Theorem 4.59. *A simple graph $G = (V, E)$ is connected if and only if it has a spanning tree.*

Proof. If G has a spanning tree, it is connected, by definition of a tree.

If G is connected, consider a connected subgraph $H \subseteq G$ with vertex set V and with the smallest possible number of edges. If H contains a cycle, we can remove an edge and it will remain connected, with the same vertex set V, contradicting our assumption that H had the smallest possible number of edges. So H must be a spanning tree. □

Equipped with this, we can now prove the converse of Theorem 4.54.

Theorem 4.60. *Given a connected simple graph $G = (V, E)$, if $|E| = |V| - 1$, the graph is a tree.*

Proof. We know that G has a spanning tree $T = (V, F)$ where $|F| = |V| - 1$, and $F \subseteq E$. But since $|E| = |V| - 1$, $|F| = |E|$, and so it follows that $F = E$ and hence $G = T$. □

4.5 Algorithms on graphs

We now turn to computational procedures for determining whether a graph possesses certain properties. Since the goal is to show existence, these are *decision* problems. In the next chapter, we will consider *optimization* problems, which use some but not all of the same methods.

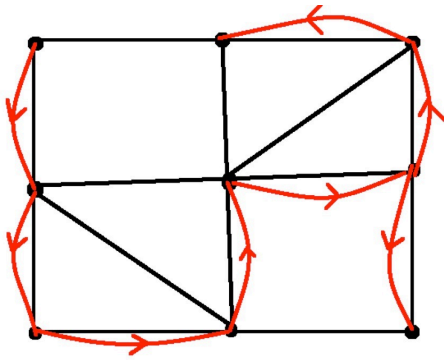
The main algorithmic technique we discuss here is the *depth-first search*, which systematically explores the vertices of a graph. This procedure involves following the branches of a tree and backtracking where necessary. While computationally it is not always the most efficient technique, it allows us to find solutions to a wide variety of problems. We consider three of them now.

4.5.1 Connectivity

We have seen in Theorem 4.59 that a simple graph is connected if and only if it has a spanning tree. So we determine whether the graph is connected by finding a spanning tree for the graph, or showing that none exists.

Algorithm

Starting from an initial vertex, we form a path by iteratively adding edges, until there is no way to do this without forming a cycle. At that point, we backtrack along the path to the previous vertex, and attempt to construct a new branch from there:



We continue in this way until we can no longer backtrack because we are back at our initial vertex. At this point, all vertices accessible from the initial vertex have been explored.

The algorithm is as follows:

```
Choose arbitrary vertex v as root of tree
Initialize list of vertices to contain v only

Repeat:
  If v has a neighbor u that is not already in the list
    Add u to list
    Let v:=u
  Else
    If v is first vertex in list, STOP
    Let v be previous vertex in list
  End
```

If the list contains all vertices, we have constructed a spanning tree, and G must be connected. If the list does not contain all vertices, there must exist a set of vertices that are not neighbors of any in the list, so G is not connected.

Complexity

Let $f(n)$ be the number of computational steps to run the algorithm, where $n = |V|$.

This depth-first search travels along each edge in the graph at most twice (once in the forward direction, and once when backtracking). Since $|E| \leq \binom{n}{2}$, $f(n) = O(n^2)$. For many graphs, $O(n^2)$ is quite reasonable. Clearly it is *not* reasonable for, say, a graph of gmail contacts, where the number of nodes is almost a billion. Fortunately, such graphs are often sparse, with vertices typically having at most a constant number of neighbors. The number of edges will only be $O(n)$ in that case, and the complexity of the algorithm will be linear.

4.5.2 Coloring

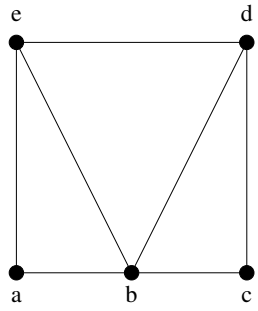
We have looked at the problem of finding the chromatic number of a simple graph G. How, algorithmically, can we test whether $\chi(G) \leq k$ for some fixed number of colors k? In certain cases, such as for planar graphs, we happen to know that there is an upper bound $\chi(G) \leq 4$ on the chromatic number. But to answer the question for general graphs, we again use depth-first search.

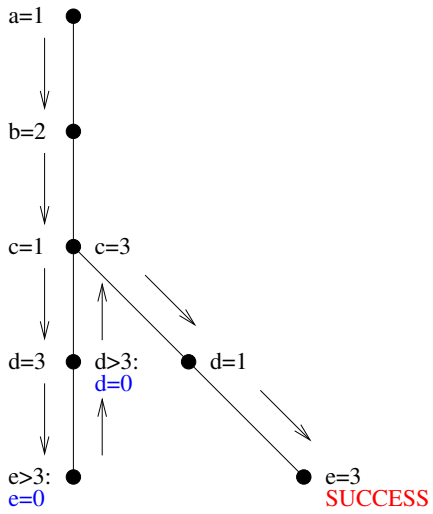
Algorithm

```
Construct ordered list of all vertices (the order is arbitrary)
Give all vertices "unassigned" color: c(v)=0 ∀v∈V
Let v be first vertex in list
Let c(v):=1

Repeat:
  If c(v)<=k
    If colors are assigned to all vertices, STOP ("SUCCESS")
    Let v be next vertex in list
    Let c(v):=min{j>0: ∀ neighbors u of v, c(u)≠j}
  Else
    Let c(v):=0
    If no colors are assigned to any vertex, STOP ("FAILURE")
    Let v be previous vertex in list
    Let c(v):=min{j>c(v): ∀ neighbors u of v, c(u)≠j}
  End
```

Take the following graph as an example, choosing vertex a as the arbitrary starting point. The accompanying tree shows the progress of the algorithm when $k = 3$.





Note that had we chosen $k < 3$, the algorithm would have backtracked beyond the starting vertex, and returned "FAILURE."

Complexity

The algorithm uses each branch of the depth-first search at most twice. But at the point where j vertices remain to be added to the list, we could need to try out all k colors for each of them. So if $n = |V|$,

$$f(n) \leq 2\sum_{j=1}^{n} k^j = O(k^n)$$

which grows exponentially.

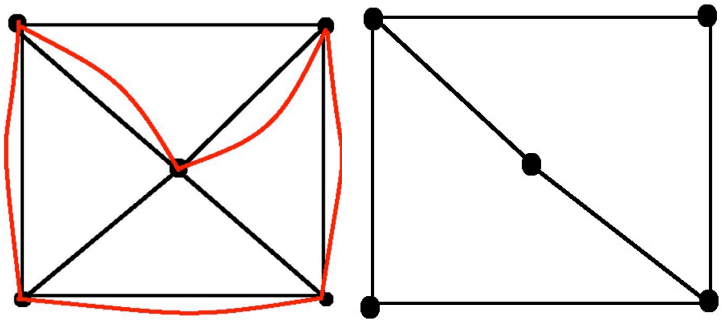
In practice, there are many cases where this algorithm is efficient. But in the worst-case scenario, we presumably cannot do much better than it, because of the inherent complexity of the problem:

Theorem 4.61. *Finding whether $\chi(G) \leq k$ is NP-complete.*

4.5.3 Hamiltonian circuit

Definition 4.62. *A Hamiltonian circuit in a simple graph $G = (V, E)$ is a path that visits each vertex in V exactly once, and returns to its starting point.*

Example 4.63.



The graph on the left has a Hamiltonian circuit, shown here. The graph on the right does not.

Hamiltonian paths and cycles are named after William Rowan Hamilton, who invented the Icosian game (now also known as Hamilton's puzzle), which involves finding a Hamiltonian circuit in the edge graph of the dodecahedron. Hamilton solved this problem using the Icosian Calculus, an algebraic structure based on roots of unity with many similarities to quaternions (also invented by Hamilton!).

Some applications of Hamiltonian circuits are in vehicle routing, facility location and coding theory.

Unlike finding a spanning tree, finding a Hamiltonian circuit can be computationally hard:

Theorem 4.64. *Finding whether an arbitrary graph has a Hamiltonian Circuit is NP-complete.*

Fortunately, there are many special cases where we can easily determine whether or not a Hamiltonian circuit exists. Here is one necessary condition:

Proposition 4.65. *If G has a Hamiltonian circuit, then $\forall v \in V, deg(v) \geq 2$*

And here is a sufficient one:

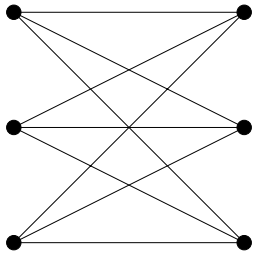
Theorem 4.66 (Ore, 1960)**.** *If $|V| > 2$ and if $\forall \{u, v\} \notin E$, $\deg(u) + \deg(v) \geq |V|$, then G has a Hamiltonian Circuit.*

The following special case is a useful corollary.

Corollary 4.67 (Dirac, 1932)**.** *If* $|V| > 2$ *and* $\forall v \in V$, $\deg(v) \geq |V|/2$, *then* G *has a Hamiltonian Circuit.*

Here is an example of a graph that satisfies the conditions of the corollary.

Example 4.68.



All vertices here have degree 3, so for any pair $\{u, v\}$ of vertices, $\deg(u) + \deg(v) = 6 = |V|$. Thus, G must have a Hamiltonian circuit.

Fortunately, while the Hamiltonian circuit problem may be computationally intractable in the worst-case scenario, it turns out to be tractable in the typical case.

Theorem 4.69 (Bollobas et al., 1987)**.** *Let* $G = (V, E)$ *be a simple graph chosen uniformly at random from all graphs with* $|V| = n$ *and* $|E| = m$, *for* n *and* m *given. Then if* G *has a Hamiltonian circuit, the circuit can be found in polynomial time with probability* $1 - o(1)$.

Note that $o(1)$ is a function that vanishes for large n. So the theorem states that with probability tending towards 1 for large graphs (*with high probability*), one can find a Hamiltonian circuit in polynomial time if one exists.

Proof. The proof is constructive, using the randomized HAM algorithm developed by Bollobas et al. to find a Hamiltonian circuit. The algorithm runs in $o\left(n^{4+\epsilon}\right)$ time for any $\epsilon > 0$. The probability of its failing to find an existing Hamiltonian circuit vanishes in the limit $n \to \infty$. □

Note that n^4 can still be quite large! But it is nowhere near as bad as, say, a function growing exponentially in n.