Introduction to Graph Theory

by Irene Lo (ilo@stanford.edu). *Based on class notes by Peter Maceli and Adrian Tang*

September 11, 2019

1 Graph Basics

A **graph** is a mathematical object we use to think about networks. It consists of a bunch of points, called **vertices**, and lines joining pairs of vertices, called **edges**. We usually let G denote the graph, V the set of vertices and E the set of edges, and write $G = (V, E)$.

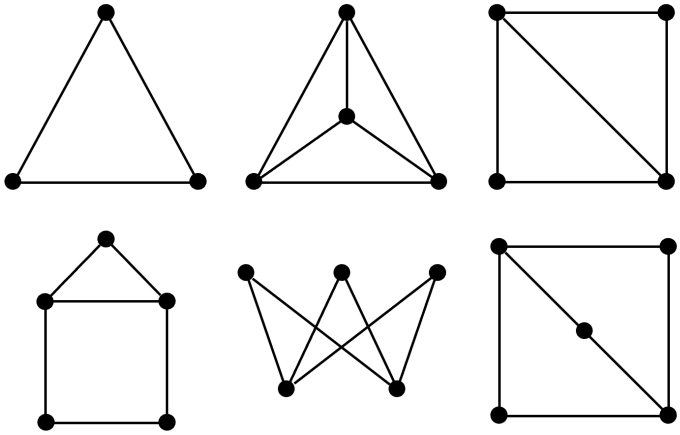


Figure 1: Some small graphs

We say two vertices are **adjacent** or are **neighbors** if they are joined by an edge, and **non-adjacent** otherwise. We say an edge and a vertex are **incident** if the vertex is one of the endpoints of the edge. The **degree** of a vertex v is the number of edges incident with v. Given a graph $G = (V, E)$, it is common to let n denote the **number of vertices** in G, and let m denote the **number of edges** in G.

Q: What is least number of edges a graph on n vertices can have? What is the most number of edges a graph on n vertices can have?

A: *The least number of edges a graph on n vertices can have is* 0. *This is achieved by an* ***independent set*** *of size n (see below). The most number of edges a graph on n vertices can have is* $\binom{n}{2} = \frac{n(n-1)}{2}$. *This is achieved by a* ***clique*** *of size n (see below).*

A **clique** in a graph G is a set of vertices where every pair of vertices is joined by an edge. An **independent set** is a set of vertices where no pair of vertices is joined by an edge.

Lemma 1 (**Handshaking Lemma**). *Let* $G = (V, E)$ *be a graph. Then*

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

Proof. We show that both expressions give us the total number of edge endpoints (i.e. the number of (vertex,edge) pairs where the vertex is incident with the edge). The left hand side counts the number of edge endpoints by first summing over the number of edges incident with each vertex ($deg(v)$ for v), and then summing over vertices. The right hand side counts the number of edge endpoints by first summing over the number of vertices incident with each edge (2 for each edge), and then summing over edges.

□

1.1 Graph Basics: Problems

Warmup Problems

1. How small/large can the degree of a vertex in a graph with n vertices be?
2. Is there a family of graphs such that every vertex has degree 0? 1? 2? 1 or 2?
3. Given integers p and q, is there a graph such that every vertex either has degree p or q?
4. Let a vertex be **even** if it has even degree, and **odd** if it has odd degree. Does every graph have an even vertex? An odd vertex? Is there a graph with exactly one even vertex? Exactly one odd vertex?
5. Let G be a graph with n vertices and m edges. Show that G has a vertex of degree at least $\lfloor \frac{2m}{n} \rfloor$, and a vertex of degree at most $\lceil \frac{2m}{n} \rceil$.

Olympiad-Style Problems:

1. (IMO Shortlist 2001) Define a k-clique to be a set of k people such that every pair of them are acquainted with each other. At a certain party, every pair of 3-cliques has at least one person in common, and there are no 5-cliques. Prove that there are two or fewer people at the party whose departure leaves no 3-clique remaining.

2 Graph Connectivity

A **walk** in a graph G is a sequence of vertices $v_0 - v_1 - v_2 - \cdots - v_\ell$ such that each vertex v_i is adjacent to the vertex v_{i-1} before it and the vertex v_{i+1} after it. We call ℓ the **length** of the walk. A **path** in a graph G is a walk where all the vertices are different.

Q: Given a walk between two vertices in a graph, how do we obtain a path between them? Is there always a walk between two vertices in a graph?

A: *Given a walk, we can obtain a path between them by removing all* ***cycles*** *in the path (see below). There is a walk between two vertices in a graph if and only if they are in the same* ***connected component*** *of the graph (see below).*

A **cycle** in a graph G is a sequence of vertices $v_0 - v_1 - v_2 - \cdots - v_\ell = v_0$ such that each vertex v_i is adjacent to the vertex v_{i-1} before it and the vertex v_{i+1} after it, where the indices are taken modulo ℓ. We call ℓ the **length** of the cycle.

A graph is **disconnected** if it can be divided into two parts with no edges between the parts, otherwise it is **connected**. A **connected component** of a graph is a connected subgraph which is as large as possible.

Q: Is there always a path between any two vertices in a connected graph? If any pair of vertices in a graph can be connected with a path, then is the graph connected?

A: *Yes, there is a path between any two vertices in a connected graph. We show this using a proof by contradiction. Suppose for the sake of contradiction that there are two vertices u, v in a connected graph G such that there is no path in G from u to v. Let A be the set of vertices that can be reached by a path from u (including u), and let $B = V \setminus A$. By assumption we have that $v \in B$, so B is not empty. Since G is connected, there must be an edge between A and B. Let us pick one such edge $e = ab$ with $a \in A$ and $b \in B$. Then since $a \in A$ there is a path P from u to a (we write $u - P - a$), and so we may append the edge e to the end to obtain a walk $u - P - a - b$ from u to b. But since there is a walk from u to b this means there is a path from u to b, which contradicts the definition of B.*

Yes, if any pair of vertices in a graph can be connected with a path, then the graph is connected. We show this using a proof by contradiction. Suppose for the sake of contradiction that there is a disconnected graph G such that every pair of vertices can be connected with a path. Let $G = (V, E)$, and since G is disconnected let $V = A \cup B$ where A, B are disjoint sets of vertices such that there is no edge between A and B. Pick arbitrary vertices $a \in A$ and $b \in B$; by assumption there is a

path P from a to b. Let $P = v_0 - v_1 - \cdots - v_\ell$, where $v_0 = a$ and $v_\ell = b$. Let k be the smallest index such that $v_k \in B$, notice that k exists since $v_\ell \in B$, and $k > 0$ since $v_0 \in A$. Then $v_{k-1}v_k$ is an edge with an endpoint $v_{k-1} \in A$ and the other endpoint $v_k \in B$, which contradicts that there are no edges between A and B.

I.e., A graph is connected if and only if there is a path between any two vertices in the graph.

2.1 Shortest Paths and Google Maps

One application of paths in graphs is finding *driving directions*. Let's say I want to go to Berkeley Bowl after class. What's one way for me to get there? What's the fastest way for me to get there?

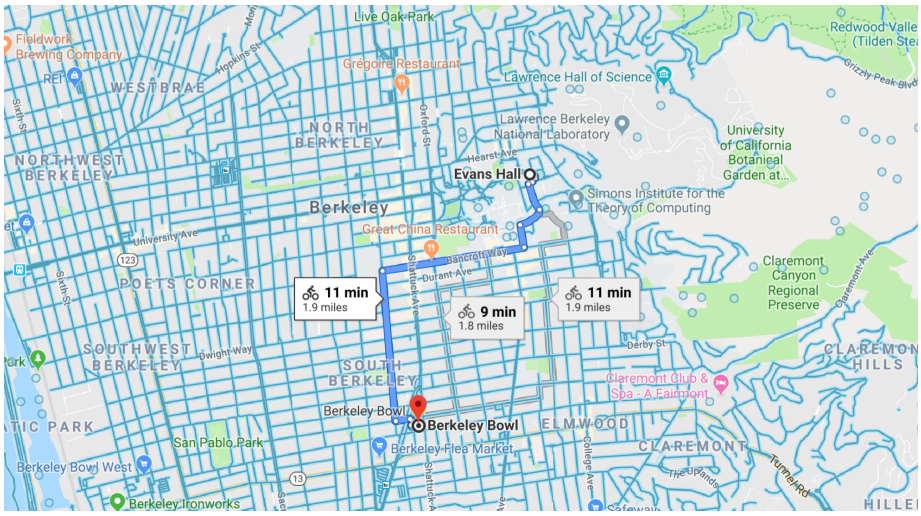


Figure 2: Google Maps from Evans Hall to Berkeley Bowl

We can formulate this problem as a graph theory problem, where $G = (V, E)$ is a graph whose vertices V are intersection points between roads in Berkeley, and edges are road segments. Then we can think of Evans Hall and Berkeley Bowl as two vertices u and v in my graph G, and my questions become the following: How can we find whether there is a path between u and v? How can we find a *shortest* path from u to v?

One possible algorithm is **depth-first search**. The idea of depth-first search is to start at u and walk as far as possible searching for v before backtracking. A simple modification of our argument for paths in connected graphs will show that depth-first search allows us to find a path from u to v, or verify that no such path exists.

Algorithm 1 (Depth-first search)**.** *Keep track of the current vertex $v_{current}$, start at $v_{current} = u$. For each visited vertex v also keep track of the 'parent' of v, $p(v)$.*

- *While $v_{current} \neq v$:*
  - *If $v_{current}$ has unvisited neighbors: Move to an unvisited neighbor w of $v_{current}$ and set $p(w) \leftarrow v_{current}$ and $v_{current} \leftarrow w$.*
  - *Else if $v_{current}$ has no unvisited neighbors:*
    - *If $v_{current} = u$, return 'There is no path from u to v'.*
    - *Move back to the parent $p(v_{current})$ of $v_{current}$ and set $v_{current} \leftarrow p(v_{current})$.*
- *If $v_{current} = v$, return the path $v - p(v) - p(p(v)) - \cdots - u$.*

Fun fact: *If you click the first (non-italicized) term of nearly any Wikipedia entry, eventually you end up at the 'Philosophy' page! In other words, we can think of Wikipedia pages as vertices in a graph and put edges between pages that have non-italicized hyperlinks between them. If we list edges from a vertex v in the order in which hyperlinks appear on page v, then* ***depth-first search on Wikipedia takes us to 'Philosophy'****.*

Another possible algorithm is **breadth-first search**. The idea of breadth-first search is to start at u and systematically search through all the vertices 'closest' to u, and then all the vertices 'second-closest' to u, and so on and so forth, where closeness is measured by the number of edges in the shortest path from u to each vertex. We call the number of edges in the shortest path from u to v the **distance** from u to v.

Algorithm 2 (Breadth-first search)**.** *Keep track of the current vertex $v_{current}$, start at $v_{current} = u$. Keep track of a queue Q of unvisited vertices in order of discovery. For each visited vertex v also keep track of the 'parent' of v, $p(v)$.*

- *While $v_{current} \neq v$:*
  - *Add all unvisited neighbors of $v_{current}$ that are not already in Q to the end of Q.*
  - *If Q is non-empty, update $v_{current}$ to the first vertex w in the queue (i.e. remove w from Q, and set $v_{current} \leftarrow w$).*
  - *Else if Q is empty, return 'There is no path from u to v'.*
- *If $v_{current} = v$, return the path $v - p(v) - p(p(v)) - \cdots - u$.*

We can show that breadth-first search finds, for each vertex $w \in G$, a path from u to w with the shortest length.

Proof Sketch: *In the first step breadth-first search finds all vertices that are distance* 1 *from* u*, let us call this set* $N(u)$ *(these are the neighbors of* u*). Each vertex distance* 2 *from* u *(let us call this set* $N_2(u)$*) can be connected to* u *by first finding a path of length* 1 *to a vertex in* $N(u)$ *and adding one more edge, and similarly for all* k *each vertex distance* k *from* u *(let us call this set* $N_k(u)$*) can be connected to* u *by first finding a path of length* $k-1$ *to a vertex in* $N_{k-1}(u)$ *and adding one more edge. Hence by using breadth-first search we can make sure to visit first all the vertices in* $N(u)$*, then visit all their unvisited neighbors (which is* $N_2(u)$*), then visit all the unvisited neighbors of vertices in* $N_2(u)$ *(which is* $N_3(u)$*), so on and so forth, and so visit all the vertices in order of their distance from* u*.*

If you've been thinking about how well my graph theory formulation models my grocery run problem, you might not be so convinced that breadth-first search will give me what I want. This is because a shortest path in my model is a path with the least *number* of edges. But what I really care about is the *time* it takes to traverse the edges. To capture this, I will need the concept of *weighted* edges.

A **weighted graph** is a graph where each edge is assigned a number, called its **weight**. For an edge e we let $\omega(e)$ denote its weight.

Algorithm 3 (Dijkstra's Algorithm)**.** *Keep track of the current vertex* $v_{current}$*, start at* $v_{current} = u$*. Keep track of a set* U *of unvisited vertices, and for each keep track of an estimated 'distance' from* u *to* v*,* $d(v)$*. For each visited vertex* v *also keep track of the 'parent' of* v*,* $p(v)$*, and the actual distance from* u *to* v*,* $d^*(v)$*. Set* $d^*(u) = 0$*.*

- *While* $v_{current} \neq v$*:*
  - *Add all unvisited neighbors* w *of* $v_{current}$ *to* U*, estimate the distance from* u *to* w *by* $\hat{d} = d(v_{current}) + \omega(v_{current}w)$*. If* w *does not have an estimated distance from* u *or the estimated distance of* w *from* u *is larger than* $\hat{d}$*, then set* $d(w) \leftarrow \hat{d}$*.*
  - *If* U *is non-empty, update* $v_{current}$ *to the vertex* w *with the smallest estimated distance* $d(w)$ *from* u *and let the actual distance be the estimated distance (i.e. remove* w *from* U*, set* $v_{current} \leftarrow w$ *and* $d^*(w) \rightarrow d(w)$*).*
  - *Else if* U *is empty, return 'There is no path from* u *to* v*'.*
- *If* $v_{current} = v$*, return the path* $v - p(v) - p(p(v)) - \cdots - u$*.*

We can show that **Dijkstra's Algorithm** finds, for each weighted graph G and pair of vertices u, v in G, the shortest (minimum weight) path from u to v. The idea behind Dijkstra's algorithm is again to systematically search through all vertices in order of how 'close' they are to u. However, now that we have edge weights, breadth-first search no longer gives me the vertices in order of the (weighted) distance from u (can you think of a graph where this failure occurs?), and we have to *estimate* the (weighted) distance from u to each vertex v.

Graphs in Real Life: If you're interested in how Google Maps finds driving directions, you can look up the A^* algorithm. The A^* algorithm is like Dijkstra's algorithm, but uses a cleverer estimate (called a *heuristic*) of the distance from your origin vertex u to each possible destination vertex v. Google Maps likely uses a bi-directional version of the A^* algorithm (simultaneously looking for shortest paths from u to v and v to u) with other tricks (e.g. pruning, shortcuts, caching) to find the shortest path from your origin to your destination.

2.2 Cut Vertices and Edges

Let G be a connected graph. A **cut vertex** in G is a vertex whose removal disconnects the graph. A **cut edge** in G is an edge whose remove disconnects the graph.

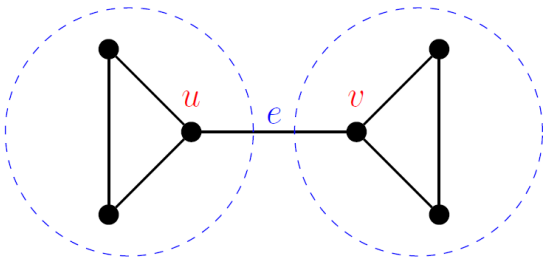


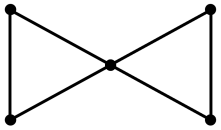
Figure 3: A graph with a cut edge

Q: If a graph has a cut edge, does it have a cut vertex? If a graph has a cut vertex, does it have a cut edge?

A: *If a graph G has a cut edge $e = uv$ and G has at least 3 vertices, then at least one of u and*

v is a cut vertex. This is because if e is a cut edge then removing e separates A from B for some disjoint sets of vertices A, B with $u \in A$ and $v \in B$. Hence removing u separates $A \setminus \{u\}$ from B, and removing v separates A from $B \setminus \{v\}$, and if G has at least 3 vertices then at least one of $A \setminus \{u\}$ and $B \setminus \{v\}$ is nonempty.

If a graph G has a cut vertex, it does not necessarily have a cut edge. E.g.



Q: When is an edge a cut edge?

Lemma 2. *Let G be a connected graph. An edge in G is a cut edge if and only if it does not lie in any cycle of G.*

Proof. Let $G = (V, E)$ be a graph, let $e = uv$ be an edge, and let $G \setminus e$ be the graph obtained from G by removing the edge e. Let A be the set of vertices that can be reached by a path in $G \setminus e$ from u (including u), and let B be the set of vertices that can be reached by a path in $G \setminus e$ from v (including v).
We show first that $A \cup B = V$. For suppose there is some vertex $w \in V \setminus A$. Since G is connected there is a path P from u to w in G, and since $w \notin A$ this means that P is not in $G \setminus e$, i.e. e is an edge in P. But then $P \setminus e$ is a path from v to w in G and so $w \in B$.
Let $G[A]$ be the graph with vertices A and all edges in $E \setminus e$ between vertices in A, and similarly define $G[B]$. It is easy to see that $G[A]$ and $G[B]$ are both connected. Hence e is a cut edge if and only if $G \setminus e$ is disconnected, which occurs if and only if the components of $G \setminus e$ containing u and v are separate components of $G \setminus e$, i.e. $u \notin B$, $v \notin A$ and $A \neq B$. We show that this occurs if and only if e does not lie in any cycle of G.
Suppose e lies in some cycle of G. Then there is a path from u to v in the graph $G \setminus e$, and so $v \in A$ and $A = B$. Hence e is not a cut edge.
Suppose e does not lie in any cycle of G. Suppose for the sake of contradiction that $A = B$. Then there is a path P from u to v in the graph $G \setminus e$, and so $P \cup e$ is a cycle, which is a contradiction. Hence $A \neq B$ and e is a cut edge.
□

2.3 Graph Connectivity: Problems

Warmup Problems

1. How many components can a graph on n vertices have?

2. Can a connected graph have more vertices than edges?
3. Give an example of a graph with no cut vertices and no cut edges.
4. If every vertex in a connected graph has even degree, can there be a cut vertex? A cut edge?

Harder Problems

1. Let G be a graph with $n \geq 2$ vertices such that every vertex has degree at least $\frac{n-1}{2}$. Show that G is connected.
2. Let G be a connected graph. Prove that two paths which are both a longest path in the graph contain at least one vertex in common.
3. Let G be a connected graph with an even number of vertices. Show that you can select a subset of edges of G such that each vertex is incident with an odd number of selected edges.
4. A graph G is **bipartite** if its vertices can be colored either red or blue such that no pair of adjacent vertices are the same color. Show that G is bipartite if and only if it does not contain a cycle of odd length.

Olympiad-Style Problems

1. (Italy 2007) Let n be a positive odd integer. There are n computers and exactly one cable joining each pair of computers. You are to color the computers and cables such that no two computers have the same color, no two cables joined to a common computer have the same color, and no computer is assigned the same color as any cable joined to it. Prove that this can be done using n colors.

3 Trees

Q: What is the smallest number of edges a connected graph on n vertices can have?

If you try to draw connected graphs with the fewest number of edges possible, you'll start to notice that they have a particular structure. In this section, we'll describe this structure.

A graph is **acyclic** if it contains no cycles. We call an acyclic graph a **forest**, and a connected acyclic graph a **tree**. A **leaf** is a vertex in a tree with degree one.

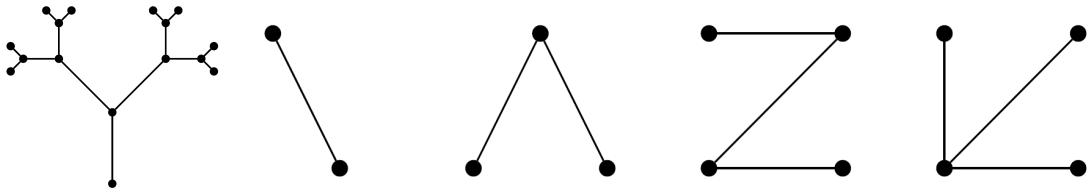


Figure 4: Some examples of trees

Q: How many edges can a tree have?

A: *A tree on n vertices has exactly $n-1$ edges. We first show that removing an edge e from a forest with k components creates a forest with $k+1$ components. It is clear that removing an edge from a forest creates a forest (as it cannot create any new cycles). Moreover removing e does not affect any components not containing e, so we can consider just the component of the forest containing e, which is a tree. Hence it suffices to show that removing an edge from a tree disconnects the tree. But this is true since every edge in a tree is a cut edge.*

Hence by induction if we remove $n-2$ edges from a tree with n vertices then we create a forest with $n-1$ components, and so at least two vertices are still joined by an edge. If we remove $n-1$ edges from a tree with n vertices then we create a forest with n components, and so there are no more edges. Hence the tree has exactly $n-1$ edges.

In fact we can use the same argument to show the following generalization:

Lemma 3. *A forest with n vertices and k components contains $n-k$ edges.*

In the above, we considered what happened when we removed edges from a tree.

Q: What happens when you add an edge to a tree?

A: *Adding an edge $e = uv$ to a tree creates a unique cycle, given by $u - P - v - u$, where P is the unique path in the tree from u to v (see if you can prove that this path is unique!).*

3.1 Spanning Trees and Electrical Grids

A **subgraph** of a graph G is a graph obtained by deleting edges and vertices from G. A **spanning tree** of a graph G is a subgraph of G that is a tree containing all the vertices of G.

Lemma 4. *A graph is connected if and only if it has a spanning tree.*

Proof. Suppose G has a spanning tree. Then G is connected, since for every pair of vertices u, v there is a path from u to v, namely the path in the spanning tree.
Suppose G is connected. Let $e_1, e_2, \ldots, e_m$ be the edges of G in some order. If we go through the edges of G in order and remove edge e_i if e_i is in a cycle in $G \setminus \{e_1, \ldots, e_{i-1}\}$ then we end up with a graph G'. It is easy to check that G' is connected (if we remove e_i then since e_i is in a cycle the remaining graph is still connected), and that G' has no cycles (if i is the largest index such that e_i is in a cycle of G', then e_i is in a cycle in $G \setminus \{e_1, \ldots, e_{i-1}\}$ and would have been removed), and so G' is a spanning tree of G.

□

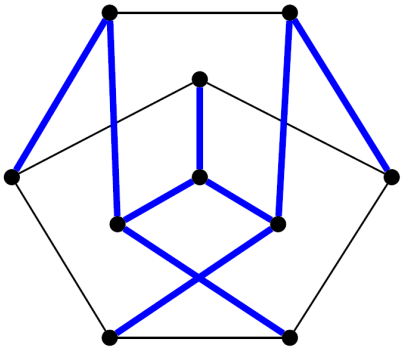


Figure 5: A spanning tree of the Petersen graph

One application of spanning trees in graphs is creating *electricity distribution networks*. Let's say I want to build electricity lines between cities in the US to make sure that every city has power. How can I do this?

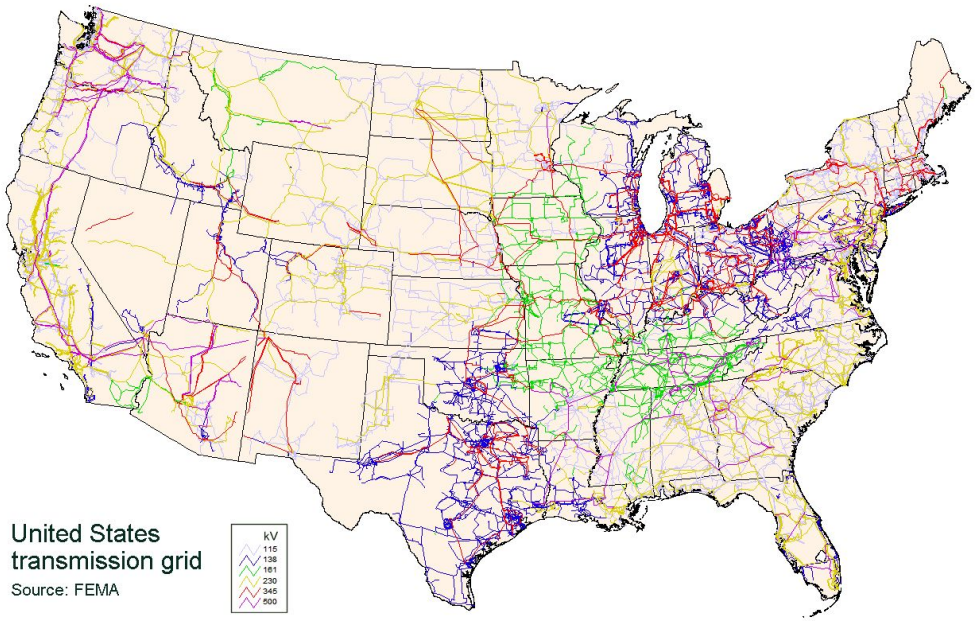


Figure 6: USA electricity transmission grid

We can formulate this problem as a graph theory problem, where $G = (V, E)$ is a graph whose vertices V are cities, and edges are pairs of cities between which we can build a power line. Then my question becomes the following: How can we find a spanning tree in G? If we add weights for

how costly it is to build each edge, we can also ask: What is the minimum-weight spanning tree in G?

Two simple algorithms that find a minimum-weight spanning tree are **Kruskal's Algorithm** and **Prim's Algorithm**. The idea of both algorithms is to add edges in increasing order of weight. Kruskal's algorithm considers all edges in order, and adds only the ones which do not create a cycle. Prim's algorithm starts from a given vertex u and grows a spanning tree from u by considering only edges that connect the spanning tree so far with the disjoint vertices not yet added to the tree.

Algorithm 4 (Kruskal's Algorithm)**.** *Start with edges $E' = \emptyset$, sort edges in order of increasing weight. For each e edge in E (considered in order of increasing weight):*

- *If there is no cycle in $G' = (V, E' \cup \{e\})$, add e to E' and update $E' \leftarrow E' \cup \{e\}$.*

Algorithm 5 (Prim's Algorithm)**.** *Start with edges $E' = \emptyset$, vertices $V' = \{v\}$. While there is a minimum weight edge e from V' to $V \setminus V'$:*

- *Remove e from E and update $E \leftarrow E \setminus \{e\}$.*
- *If there is no cycle in $G' = (V, E' \cup \{e\})$, add e to E' and update $E' \leftarrow E' \cup \{e\}$.*

3.2 Application: Hydrocarbons

Here's an interesting application of graphs from chemistry. **Hydrocarbons** are chemical compounds consisting of hydrogen atoms (H) and carbon atoms (C). Hydrogen atoms have 1 valence electron and can form a single bond with other atoms. Carbon atoms have 4 valence electrons and can form 4 bonds with other atoms. **Saturated** hydrocarbons are hydrocarbons with no cycles. (This also means there are no 2-cycles, and all bonds between pairs of atoms are single bonds.)

We can represent chemical compounds using their **bond graph**, which is a graph whose vertices are atoms and edges are bonds between two atoms.

Q: Can we use the language of graph theory to describe saturated hydrocarbons?

A: *Trees with vertices of degree only one or four!*

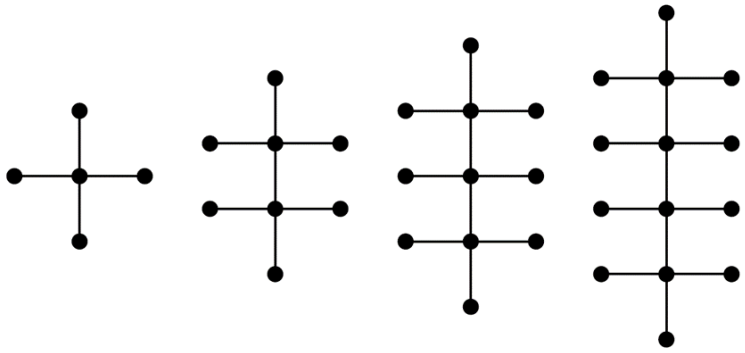


Figure 7: Some saturated hydrocarbons: methane (CH_4), ethane (C_2H_6), propane (C_3H_8), butane (C_4H_{10})

In 1875 Arthur Cayley used graph theory to predict the existence of the following alkanes:

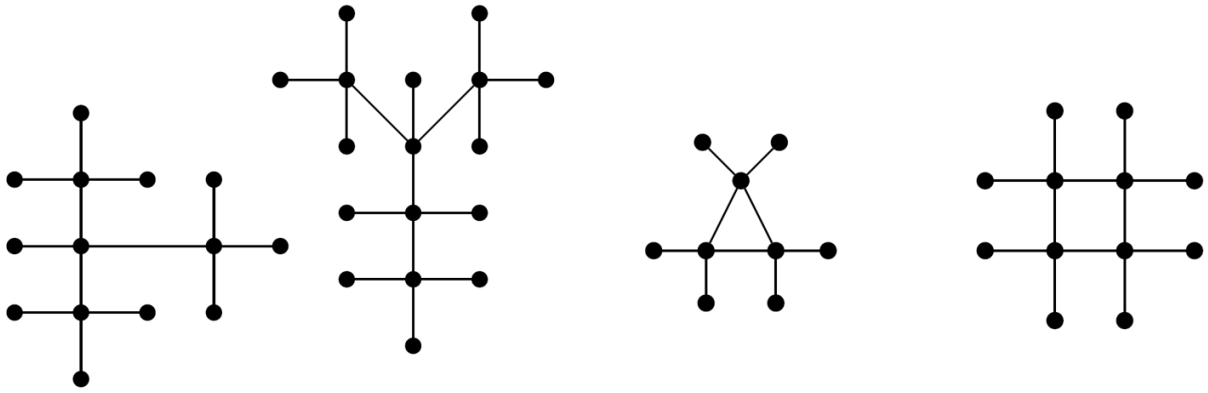


Figure 8: Some saturated hydrocarbons: isobutane (C_4H_{10}), isopentane (C_5H_{12}), cyclopropane (C_3H_6), cyclobutane (C_4H_8)

3.3 Trees: Problems

Warmup Problems

1. How many spanning trees does a tree have? How many spanning trees does a cycle have? How many spanning trees does a complete graph have?

Harder Problems

1. Show that the Petersen graph has 2000 spanning trees.
2. Let G be a tree with n vertices with maximum degree Δ. Prove that there exists an edge in G whose removal leaves two trees with at least $\lceil \frac{n-2}{\Delta} \rceil$ edges each.

Olympiad-Style Problems

1. (USAMO 2007) An animal with n cells is a connected figure consisting of n equal-sized cells. A dinosaur is an animal with at least 2007 cells. It is said to be primitive if its cells cannot be partitioned into two or more dinosaurs. Find with proof the maximum number of cells in a primitive dinosaur.
2. (Iran 2005) A simple polygon is one where the perimeter of the polygon does not intersect itself (but is not necessarily convex). Prove that a simple polygon P contains a diagonal which is completely inside P such that the diagonal divides the perimeter into two parts both containing at least $\frac{n}{3} - 1$ vertices. (Do not count the vertices which are endpoints of the diagonal.)
3. (Cayley's Formula) A labelled tree on n vertices is a tree on n vertices where all vertices are given distinct labels from 1 to n. Prove that there are n^{n-2} labelled trees on n vertices. *(This is hard to show but has many beautiful proofs – look up 'proofs from the book'!)*