
Graph theory

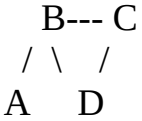
1. Basic definitions
2. Isomorphic graphs
3. Representation of graphs
4. Paths
5. Eulerian and Hamiltonian graphs
6. Longest and shortest paths
7. Colorability

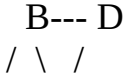
1. Basic Definitions

Def: A (finite, simple) **graph** is an ordered pair $G = (V, E)$ where
- V is a nonempty finite set (the *vertices* or *nodes*)
- E is a set of two-elements subsets of V (the *edges*)

There are many other "kinds" of graphs—for example, in a **directed graph** (**digraph**), the edges (now often called **arcs**) are **ordered** pairs, instead of unordered pairs. We sometimes allow graphs with **self-loops** (and edge between a vertex and itself) or **multiple edges** (two or more different edges connecting a pair of nodes). In a **network**, each edge (or arcs) has a real-valued **weight**. People also consider **infinite graphs**. We even have graphs where an edge can be **incident** (touch) touch more than two vertices (these are called **hypergraphs**). None of these variants are not allowed in simple graphs.

Conventional representation: a **picture**. (Draw some.) But be clear: the picture is **NOT** the graph, it is a **representation** of the graph. The graph is the pair (V, E).





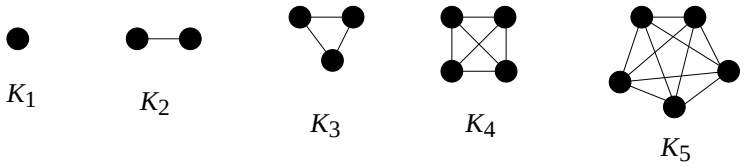
A C

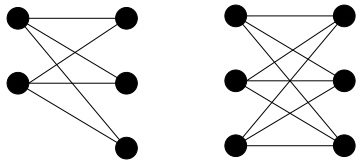
are the SAME graph.

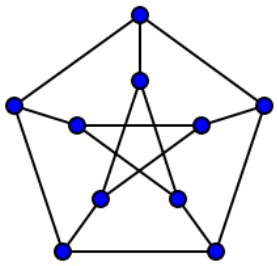
Not a graph: draw a self-loop, multiple loops, empty set.

Graphs are interesting in part for there versatility to model other phenomena. Often the vertices represent "things" and an edge between them means that they stand in relation to one another. So graphs directly model symmetric, anti-reflexive relations. Things like V for a set of People and E for the binary relation is-friends-with. Or: V for buildings and E for is-adjacent-to. Or V for countries and E for shares-a-border. And so on…

Some "special" graphs – a **clique** of size n, K_n . The complete **bipartite graphs** on n "boys" and m "girls", $K_{n,m}$.







The Peterson graph

Def: Vertices v, w of a graph $G=(V, E)$ are **adjacent** if $\{v,w\} \in E$.
The **neighbor set** of a vertex v, $N(v)$, is $\{w \in V: \{v,w\} \in E\}$.
The degree of a vertex, deg(v), is its number of neighbors.
So deg(v) = $|N(v)|$.

Question: what is the maximal and minimal degrees that one might have in an n-vertex graph?

I like $e=\{x,y\}$ for an edge, emphasizing that the vertices **incident** to an edge are **unordered**. But one will sometimes see xy or (x,y). Such notation is potentially misleading because it looks like the order matters, which, in a simple graph, is not the case.

Usually we use $n=|V|$ and $m=|E|$; alternatively, $\nu = |V|$ and $\varepsilon = |E|$.

Proposition: In a graph $G=(V,E)$ with m edges, $\Sigma_v \deg(v) = 2m$.

We don't usually care about the *names* of points in V, only how they're connected up. That's because the properties of the graphs that matter are those that are invariant under isomorphism.

2. Isomorphic graphs

Def: Graphs $G=(V, E)$ and $G'=(V', E')$ are **isomorphic** if there is a permutation π: $V \to V'$ such that $\{v,w\} \in E$ iff $\{\pi(v),\pi(w)\} \in E'$.

Claim: Isomorphism is an equivalence relation.

Give examples of properties that are and are not preserved under isomorphism.

Graphs are usually considered "the same" if they are isomorphic. One consequence of this is that there is usually no significance ascribed to the names of the vertices—the actual values in the set V. But the number of vertices matters, of course. So we can often just assume that $V=\{1,...,n\}$ for some number n.

Amazing fact: there is no efficient algorithm known to decide if two graphs are isomorphic. Most computer scientists believe that no such algorithm exists. But this is one of the biggest open questions in theoretical computer science.

How many different graphs are there on $V=\{1,...,n\}$? $2^{C(n,2)} = 2^{n(n-1)/2}$

It is harder to count how many non-isomorphic graphs there are. Maybe do this for n=3, n=4.

3. Representation of graphs

There are two common ways to represent a graph in a computer: an *adjacency matrix* and an *adjacency list*.

Adjacency matrix: An n x n matrix binary matric where $A_{ij} = 1$ if $\{i, j\} \in E$ and $A_{ij} = 1$ otherwise.

Adjacency list: An n-element list of lists, each of which is itself a list having at most n-1 elements. The i-th entry in the list is the list of all vertices adjacent to vertex i.

This will have been covered in many students second programming class.

4. Paths

Def: A **path** $p = (v_1, ..., v_n)$ in $G = (V,E)$ is a sequence of vertices s.t. $\{v_i,v_{i+1}\} \in E$ for all i in $\{1,..., n-1\}$.

A path is said to *contain* the vertices and to *contain* the edges $\{v_i,v_{i+1}\}$.

The ***length*** of a path is the number of edges on it.

A **cycle** is a path of length three or more that starts and ends at the same vertex and includes no repeated edges.

A graph is *acyclic* if it contains no cycle.

A graph $G = (V,E)$ is *connected* if, for all $x,y \in V$, there is a path from x to y.

Graph $G' = (V',E')$ is a *subgraph* of graph $G = (V,E)$ if $V' \subseteq V$ and $E' \subseteq E$.)

The *components* of a graph are the maximal connected *subgraphs*.

Alternative definition of components: Say that $x \sim y$ (these vertices are in the same component) if there is a path from x to y. **Prop**: this is an equivalence relation. Its blocks (equivalence classes) are the components.

Alternative definition of a component: the component containing v is all vertices connected to v by paths of any lengths; and all the induced edges (the edges of the original graph that span vertices in the component).

Describe an algorithm, based on DFS, for counting the number of components of a graph and identifying them.

Def: A *tree* is a connected acyclic graph.

Proposition: In any tree with n vertices and m edges, $m = n - 1$

Proof: By induction on n. True when n=1. Now suppose G is an n-vertex tree, n>1. Claim that there is some node of degree 1 in G.

If any node of degree 0: contradicts connected.

If all nodes of degree ≥ 2, contradict acyclicity.

Take your node of degree 1 and remove it, along with the adjoining edge.
What remains is connected and acyclic, so (m-1) = (n-1) -1 for it.
Took away one edge and one vertex, so m=n-1.

5. Eulerian and Hamiltonian graphs

Def: A graph G is *Eulerian* if it there is a cycle C in G that goes through every **edge** exactly once. A graph G is *Hamiltonian* if there is a cycle that goes through every **vertex** exactly once.

Theorem: (Euler) A connected graph $G = (V,E)$ on $n \geq 3$ vertices is Eulerian Iff every vertex of G is of even degree.

Proof: ➔ Choose s. Graph is Eulerian mean there is a path that starts at s and eventually ends at s, hitting every edge. Put a label of 0 on every vertex. Now, follow the path. Every time we exit a vertex, increment the label. Every time we enter a vertex, increment the label.
At end of traversing the graph, label(v) = deg(v) and this is even.
← (sketch) If every vertex is of even degree, at least three vertices. Start at s and grow a cycle C of unexplored edges until you wind up back at s.
You never "get stuck" by even-degree constraint. If every edge explored:

Done. Otherwise, find contact point of C and an unexplored edge (exists by connectedness) and grow out from there. Splice together the paths. END

So there is a trivial algorithm to decide if G is Eulerian: just check if all its vertices are of even degree.

Amazing fact: There is no "reasonable" algorithm known to decide if a graph is Hamiltonian.

(Most computer scientists believe that no such algorithm exists.)

Lots of graph problems are this way: some problem is computationally easy to solve, but some problem that might sound very similar to it is, apparently, computationally intractable. Here are a few more examples:

6. Longest and shortest paths

Def: A *shortest path* between two vertices x and y is a path from x to y such that there is no shorter (=fewer edges) path from x to y.

A *longest path* between two vertices x and y is a *simple path* (= no repeated *vertices*) from x to y.

Claim: There is an efficient algorithm to identify a shortest path between two designated vertices in a graph. (You will learn one in ecs122A or ecs60)

Amazing fact: There is no efficient algorithm known to find a longest path from x to y. (Most computer scientists believe that no such algorithm exists.)

Diameter of a graph = the length of a longest shortest path.
Explain how an *inability* to efficiently decide if a graph is Hamiltonian implies an *inability* to find a longest path between a designated pair of vertices: namely, there is a simple path (=no repeated vertices) of length $n-1$ between x and y (where $n = |V|$) iff $G \cup \{x, y\}$ has a HC.

7. Colorability

Def: A graph $G = (V, E)$ is ***k*-colorable** if we can paint the vertices using "colors" $\{1,...,k\}$ such that no adjacent vertices have the same color. Formally, $G = (V, E)$ is k-colorable if there exists a function c: $V \rightarrow \{1,...,k\}$ such that for all $\{v,w\} \in E$ implies $c(v) \neq c(w)$.

Def: A graph is **bipartite** if it is 2-colorable. Equivalently, a graph is bipartite if we can partition V into (V_1, V_2) such that all edges go between a vertex in V_1 and a vertex in V_2.

Proposition: A graph is bipartite iff it contains no odd-length cycle.

Proposition: There is a simple and efficient algorithm to decide if a graph G is 2-colorable / bipartite.

Proof: Initially, all vertices are *uncolored*. Now choose an uncolored vertex and color it 1. Color each all of it's neighbors 2. Color their neighbors 1. And so on. If one ever encounters adjacent vertices with the same color then output that the graph is not bipartite. I the process halts and there are more uncolored vertices, color one of them 1 and repeat. Continue until all vertices colored. I claim that this 2-colors all and only the bipartite graphs.

Complementary fact: There is no computationally reasonable algorithm known to decide if a graph is 3-colorable. Most computer scientists believe that no such algorithm exists.