# 20 Trees

Graphs can be used to model and solve many problems. Trees are special graphs. Today, we look at various properties of graphs as well as of trees.

**Party problem.** Suppose we choose six people at random from a party. We call the people $A$ through $F$. Then, one of the following must be true:

I. three of the people mutually know each other; or

II. three of the people mutually do not know each other.

We reformulate this claim using a graph representing the situation. We draw six vertices, one for each person, and we draw an edge between two vertices if the two people know each other. We call this a *simple graph*. The *complement graph* consists of the same six vertices but contains an edge between two vertices iff the graph does not have an edge between them. Property I says the graph contains a triangle and Property II says the complement graph contains a triangle. In Figure 24, we see two graphs on six vertices. On the left, the graph contains a triangle and on the right, the complement graph contains a triangle. We can now reformulate the above claim.

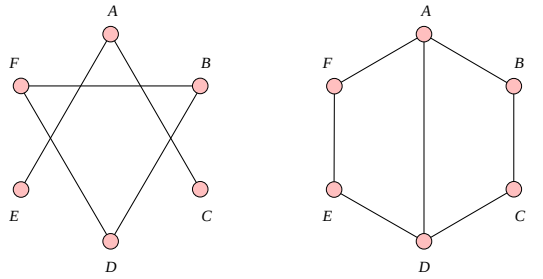


Figure 24: The two cases we consider in the party problem.

PARTY THEOREM. If a simple graph on six vertices does not have a triangle then the complement graph on the same six vertices has a triangle.

PROOF. We distinguish the case in which $A$ knows two or fewer people from the case in which $A$ knows three or more people. For the first case, we assume that $A$ possibly knows $C$ and $E$ but nobody else. If $A$ knows both and $C$ and $E$ know each other, then we have a triangle. Otherwise, consider $B, D, F$. If they do not all mutually know each other, then without loss of generality, we can say that $B$ does not know $D$. Thus, we have a triangle in the complement graph since $A$, $B$, and $D$ mutually do not know each other. For the second case, assume that $A$ knows $B$, $D$, $F$, and possibly other people. If any two of the three know each other then we have a triangle in the graph. Otherwise, we have a triangle in the complement graph since $B$, $D$, and $F$ mutually do not know each other. □

**Simple graphs.** In the above problem, we used graphs to help us find the solution. A *(simple) graph*, $G = (V, E)$, is a finite set of vertices, $V$, together with a set of edges, $E$, where an edge is an unordered pair of vertices. Two vertices connected by an edge are *adjacent* and they are *neighbors* of each other. Furthermore, we say the edge is *incident* to the vertices it connects. Sometimes we refer to the vertices as nodes and the edges as arcs. For example, Figure 25 shows the *complete graph* of five vertices, which we denote as $K_5$. This graph is complete since we cannot add any more edges. A *subgraph* of a graph $G = (V, E)$ is

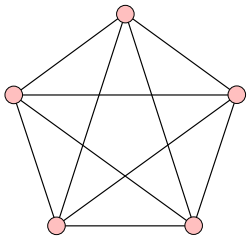

Figure 25: The complete graph of five vertices. It has $\binom{5}{2} = 10$ edges.

a graph $H = (W, F)$ for which $W \subseteq V$ and $F \subseteq E$. For example, a *clique* in $G$ is a subgraph that is a complete graph if considered by itself. With this terminology, the Party Theorem says that a graph on six vertices contains a clique of three vertices or its complement graph contains such a clique.

**Connectivity.** Suppose we have a graph where the nodes are cities and an edge $\{a, b\}$ exists if there is a train that goes between these two cities. Where can you go if you start at a city $x$? We need some definitions to study this question. A *walk* is an alternating sequence of vertices and edges such that

1. the sequence starts and ends with a vertex;
2. each edge connects the vertex that precedes the edge with the vertex that succeeds the edge.

Furthermore, a *path* is a walk such that no vertex appears twice. If there exists a walk from $a$ to $b$, then we know that there also exists a path from $a$ to $b$. Indeed, if a vertex $x$ appears twice, we can shorten the walk by removing all edges and vertices between the two copies as well as one copy of the vertex $x$. If the walk is finite, we get a path after finitely many operations as described.

CLAIM. Having a connecting path is an equivalence relation on the vertices of a graph.

PROOF. Let $a, b, c$ be three vertices of a graph $G$. The relation is reflexive because $(a)$ is a path from $a$ to $a$. The relation is symmetric because reversing a path from $a$ to $b$ gives a path from $b$ to $a$. Finally, the relation is transitive because concatenating a path from $a$ to $b$ with a path from $b$ to $c$ gives a path from $a$ to $c$. □

A graph is *connected* if there is a path between every pair of its vertices. A *connected component* of a not necessarily connected graph is a maximal connected subgraph. Equivalently, a connected component is an equivalence class of vertices together with the edges between them.

**Cycles and trees.** A *closed walk* is a walk that starts and ends at the same vertex. A *cycle* is a closed walk in which no vertices are repeated. Alternatively, we can say that a cycle is a path in which the start and the end vertices are the same. A *tree* is a connected graph that does not have any cycles.

PROPERTIES OF TREES. If $T = (V, E)$ is a tree with $n$ vertices and $m$ edges, then we have the following properties:

1. there is exactly one path between every pair of vertices;
2. removing an edge disconnects the tree;
3. the number of edges is $m = n - 1$;
4. there exists at least one vertex that has precisely one neighboring vertex.

**Spanning trees.** Sometimes the whole graph gives us more information than we need or can process. In such a case, we may reduce the graph while preserving the property of interest. For example, if we are interested in connectivity, we may remove as many edges as we can while preserving the connectivity of the graph. This leads to the concept of a *spanning tree*, that is, a subgraph that contains all vertices and is itself a tree.

SPANNING TREE THEOREM. Every finite connected graph contains a spanning tree.

PROOF. If there is a cycle in the graph, remove one edge of the cycle. Repeat until no cycles remain. □

There are a number of different algorithms for constructing a spanning tree. We may, for example, begin with a vertex $u_0 \in V$ and grow a tree by adding an edge and a vertex in each round. This is called Prim's Algorithm.

$W = \{u_0\}$; $F = \emptyset$; $X = V - \{u_0\}$;
`while` $\exists$ edge $\{w, x\}$ with $w \in W$ and $x \in X$ `do`
  move $x$ from $X$ to $W$; add $\{w, x\}$ to $F$
`endwhile`;
`if` $V = W$ `then` $(W, F)$ is spanning tree
  `else` $(V, E)$ is not connected
`endif`.

At the end of this algorithm, we have determined if $G$ is connected. If it is connected, we have found a spanning tree. Otherwise, $(W, F)$ is a spanning tree of the connected component that contains $u_0$.

**Rooted trees.** In many situations, it is convenient to have the edges of a tree directed, from one endpoint to the other. If we have an edge from $a$ to $b$, we call $a$ a *parent* of $b$ and $b$ a *child* of $a$. A particularly important such structure is obtained if the edges are directed such that each vertex has at most one parent. In a tree, the number of edges is one less than the number of vertices. This implies that each vertex has exactly one parent, except for one, the *root*, which has no parent. Holding the root and letting gravity work on the rest of the graph, we get a picture like in Figure 26 in which the root is drawn at the top. The directions of the edges are now determined, namely from top to bottom, leading away from the root. Each maximal directed path starts at the root and ends at a *leaf*, that is, a vertex without children. Rooted trees are often used to model or to support a search process. We start at the root and choose an outgoing edge to direct the search to one of the available subtrees. We then recurse, treating the new vertex like the root. Repeating this step, we eventually arrive at a leaf of the rooted tree. Similarly, we can give a recursive definition of a rooted tree. We phrase this definition of the case of a *binary tree*, that is, a rooted tree in which every vertex has at most two children. We thus talk about a left child and a right child.

- an empty graph is a binary tree;

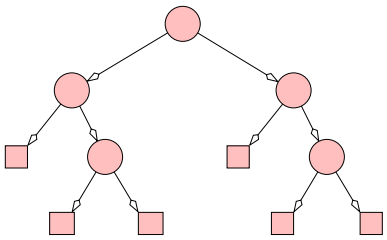

Figure 26: The top vertex is the root and the square vertices are the leaves of the rooted tree.

- a vertex (the root) with a binary tree as left subtree and a binary tree as right subtree is a binary tree.

While uncommon in Mathematics, recursive definitions are suggestive of algorithms and thus common in Computer Science.

**Summary.** Today, we looked at graphs, subgraphs, trees, and rooted trees. We used Prim's algorithm to find a spanning tree (if one exists).