Learning objectives:

- ☐ describe properties of graphs: walks, paths, cycles
- ☐ prove the number of edges of a tree is equal to the number of vertices - 1
- ☐ prove that every graph has a spanning tree
- ☐ build a spanning tree from an arbitrary connected graph

Trees are one of the most useful data structures in computer science. They have applications to image segmentation, circuit design, feature extraction, gene expression, computer games and computer-aided design. We will also see trees later in the course when we study recurrence relations and probability. The goal of this lecture is to introduce you to various tree terminology and to prove certain properties about trees. As usual, here's a puzzle.

For more applications, see here.

Example 1:
Suppose you are given eight coins. Seven of the coins have an equal weight but one of them is lighter. You have a balance to weigh any two sets of coins to determine which set is heaver (if any). What is the minimum number of weighings you need to determine which coin is lighter?

So many terms!

I know, but hopefully some of them are intuitive. At the end of this section, make sure you understand the following terms:

- walk
- path
- connected
- cycle
- root node
- leaf node

1 Tree terminology

A tree is just a special type of graph. In order to understand what "special" means, we need to make a few definitions.

Definition 1. *A **walk** in a graph $G = (V, E)$ is a sequence of vertices that are connected by edges.*

For example, in the graph of Figure 1, we might have a walk:

$$c - b - a - d - h - f - e.$$

We say that this walk is of length $k = 6$ because we traversed 6 edges to get from the starting node to the end node. When the start and end nodes are the same, we obtain a closed walk.

Definition 2. *A **closed walk** is a walk which starts and ends at the same vertex.*

For example,

$$d - e - b - c - f - g - f - h - d.$$

is a closed walk of length $k = 8$.

Definition 3. *A **path** is a walk where all the vertices (or nodes) are different.*

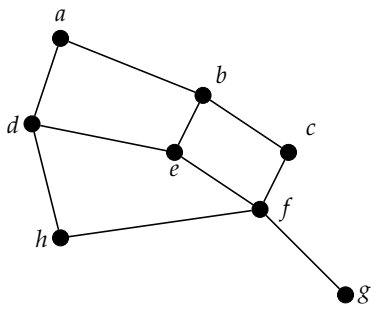


Figure 1: Example graph for defining terms.

For example, in Figure 1, $a-d-e-b-c$ is a path, but $a-d-e-b-e-f$ is not a path.

Definition 4. *A **cycle** is a closed walk of length greater than* 2 *in which all the nodes are different. A graph that does not contain any cycles is called **acyclic**.*

In Figure 1, $a-b-c-f-e-d-a$ and $e-f-h-d-e$ are a few examples of cycles.

Definition 5. *Nodes u and v are said to be **connected** if there is a path from u to v. A graph is said to be **connected** if every pair of nodes are connected.*

If a graph G is not connected, then any connected subgraphs of G are called *connected components*. Figure 2 is a graph with four connected components. Finally, we can now define a tree!

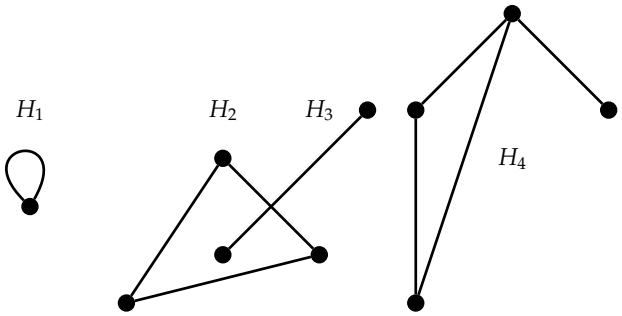


Figure 2: A single graph G with four connected components H_1, H_2, H_3 and H_4. H_2 and H_4 contain cycles. H_3 does not contain a cycle. G is not a tree because it is not connected.

Definition 6. *A **connected** and **acyclic** graph is called a **tree**.*

The graph of Figure 3 is an example of a tree. A graph that is **acyclic** but not **connected**, it is called a **forest**. A directed graph that is **acyclic** and not necessarily connected is called a **directed acyclic graph** and often abbreviated **DAG**.

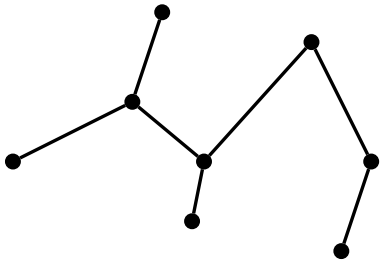


Figure 3: Example of a tree.

A nice property of trees is that we can related the number of vertices to the number of edges with the following lemma.

Lemma 1. *A tree with n vertices has $n-1$ edges.*

Proof. We use a proof by induction on the number of vertices n. Let the induction hypothesis be $p(n) =$ *there are $n-1$ edges in an n-vertex tree.*

Base case: When there is a single vertex ($n=1$), there are 0 edges, so the induction hypothesis holds.

Inductive case: Assume $p(n)$. That is, any n-vertex tree has $n-1$ edges. We need to show that any $(n+1)$-vertex tree has n-edges. Let T be a tree that has $n+1$ vertices. Let v be a **leaf** of T. Remove v from

T to get a tree T' with n vertices. By $p(n)$, T' has $n-1$ edges. Now, add v back in. We can add v anywhere we like. The key is that we can only add a single edge to create a tree (otherwise there would be cycle). After v has been added back in, then T has $(n-1)+1 = n$ edges.

By induction on the number of vertices in a tree, $p(n)$ is true. □

Why does v need to be a leaf?

Note: v needs to be a leaf of T to get a tree T', otherwise T' would not be connected and, hence, not a tree.

2 Rooted trees

By definition, the graph of Figure 3 is a tree, however, it doesn't look much like a real-life tree. If a tree has a designated root, that tree is called a **rooted tree**. See Figure 4.

Definition 7. *A **rooted tree** is a tree in which a single vertex is designated as the **root** and every edge is directed away from the root.*

Rooted trees are typically drawn in levels. A vertex connected lower than another vertex is called a **child**. The connected vertex higher than another vertex is called the **parent**. A **leaf** is a special kind of vertex that has no children. Note that the degree of a leaf is 1.

In Figure 4, the root is the parent of ℓ_1, ℓ_2 and u, whereas ℓ_3 and ℓ_4 are children of u. Nodes at the same level are called **siblings**. So ℓ_1, ℓ_2 and u are siblings, whereas ℓ_3 and ℓ_4 are siblings.

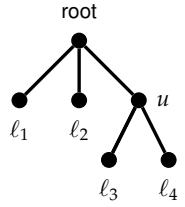


Figure 4: Example of a rooted tree with four leaves (labelled ℓ_1, ℓ_2, ℓ_3, ℓ_4).

2.1 k-ary trees

A tree in which each node has $\leq k$ children is called a k-ary tree. An examply of a 3-ary tree is in Figure 4 because there are three edges emanating from the root. We have a special name for 2-ary trees ($k = 2$): **binary trees**. Binary trees are super important in computer science - you will see them many times in the rest of your studies and career! You will learn more about them in CS 201 (Data Structures).

Determining the probability of obtaining a particular sequence of heads or tails when flipping a coin can be seen as a binary tree. Every flip of the coin branches the sequence into one of two directions. This is illustrated in Figure 5.

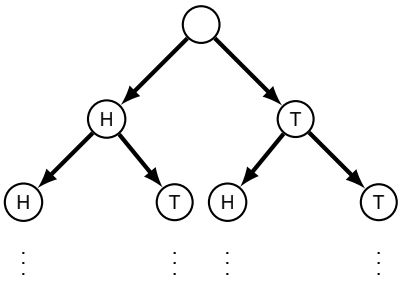


Figure 5: Binary tree representing the flipping of a coin.

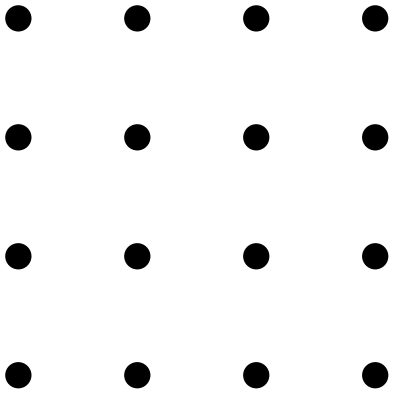
Example 2:
Can you determine the probability of obtaining the sequence H-T-T-T-H-H when flipping a fair coin?

Solution:
It doesn't really matter what the sequence is, only the number of levels matters. Assuming we have a fair coin, the probability of each branch is 0.5 (50%). Therefore, the probability of the sequence is $p = (0.5)^6 = 1.56\%$.

We'll do more examples like this when we discuss probability later in the course.

3 Spanning trees

Example 3:
Build a tree of the following set of dots such that the total length of all the edges is a minimum. You can draw an edge between *any* pair of vertices.



What we did in the last example was built a spanning tree of the set of vertices in G.

What was the original graph?

Since we said *any* pair of vertices could be connected by an edge, there was a secret hidden graph that connected every vertex to every other one. In fact, these types of graphs are called **complete**.

Definition 8. *A **spanning tree** T of a connected graph $G = (V, E)$ is a subgraph of G with the same vertices as G.*

It is important to remember that the spanning tree has exactly the same vertices as G, however, it has a subset of the edges of G. An example of a spanning tree is given in Figure 6. Note that the spanning tree is *not* unique. In particular, we could remove edge $\{b, c\}$ and add edge $\{e, f\}$ in the spanning tree on the right of Figure 6 to a get a new spanning tree.

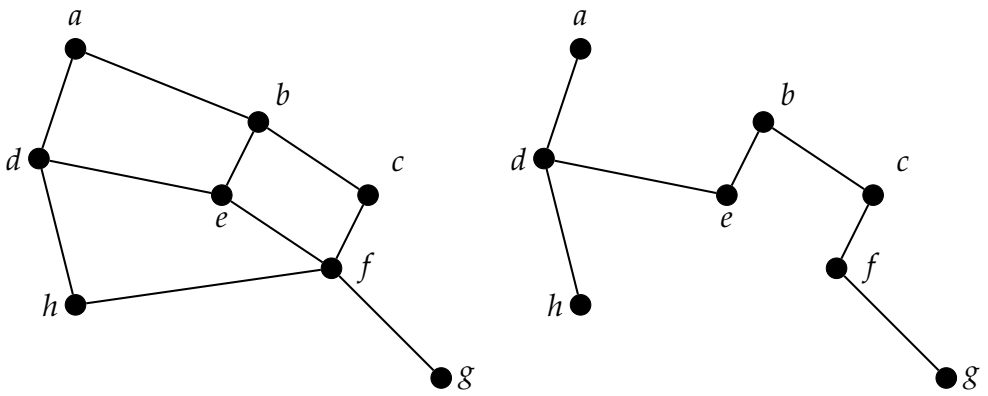


Figure 6: Example graph with a possible spanning tree.

We also have a really important result about connected graphs!

Theorem 1. *Every connected graph has a spanning tree.*

Proof. We use a proof by contradiction. Suppose a connected graph G has no spanning tree. Now, let T be a connected subgraph of G with the same vertices as G and with the *smallest* number of edges possible. Since T is not a spanning tree, then it has a cycle. But we can remove an edge from the cycle without violating connectedness. But then T would not have the smallest number of edges, so we have a contradiction. Therefore, T must be a spanning tree of G. □

Next class, we'll looking at searching algorithms, which can be used to create spanning trees.

Smallest number of edges?

We used what is called the *well-ordering principle*, which means that every non-empty set has a *least element*. Here, we are looking at the set of all possible subgraphs of G. Our least-element is the subgraph that gives us the minimum number of edges. Read more about the well-ordering principle here.