Trees with Applications to Computer Science

Chloe Carrano*

April 12, 2026

Abstract

In this work, we briefly review the foundational knowledge necessary to conceptually understand trees before delving into the study of common types of trees and their applications. Through several ventures into those varying tree-type graphs, along with a detailed proof of the Matrix-Tree Theorem, we observe the many ways that a tree graph can be represented (ex. algebraically, combinatorically) and use those many representations to understand central aspects of both graph theory and computer science.

Contents

1 Introduction

Tree structures have been in use since antiquity, first created by medieval clerics to represent potential matchmaking relationships based on societal bloodlines, then later used within religious contexts to form the famous 11th-century Tree of Jesse and show connections between the Greek classical gods. These applications were adapted for modern life as time went on – though no longer used in a matchmaking context, for example, trees became widely known as an effective method to represent *existing* relationships between family members. In the 1900s, trees had

*chloetcarrano@gmail.com

critical applications in the fields of genealogy (representing ancestry), chemistry (representing acyclic molecular structures), and taxonomy (representing hypothetical relationships between organisms within a particular classification, such as a phylum). Pioneers across multiple fields of science, such as chemist Arthur Cayley and taxonomist Charles Darwin are famous for their work connecting trees to their respective areas of study. Now, applications of trees are found in nearly every area of life – they have popular uses in GPS systems, social networking services (used to analyze user connections) and search engines. [Mul88]

The use of tree structures has been intrinsically related to the study of computer science since its birth in the 1970s, when the first personal computer was released for widespread use. One of the fundamental purposes of a computer is to store data; tree structures have helped organize data in a hierarchical way, enabling users to quickly search and sort through information. [Tos15]

Although graph theory began being formally studied in the early 1800s, trees only first appeared in mathematical studies in roughly the mid-19th century. [Chu10]

Mathematicians have been fascinated by the properties of trees ever since. Within the context of graph theory, a tree is a simple graph that is restricted by two criteria:

Definition 1.1. A **tree** is an acyclic, connected graph. [CZ12]

This paper aims to provide a background overview regarding trees before exploring more specialized topics connected to computer science such as binary search trees (trees where each edge comes from a single **root** point and branches out to form at most two smaller edges), minimum and maximum heaps (trees where every **node** has a specific value, each edge branches out to form two smaller edges, and the root is the minimum or maximum-value element), Huffman trees (trees that only store data at particular vertices and minimize the sum of those vertices' weights), prefix trees (trees where each vertex represents a particular character), and decision trees (trees where each vertex represents a particular condition or decision to be made, and each edge represents a certain outcome).

Definition 1.2. A **root** is a designated vertex on a tree graph from which there is a directed path that can reach all other vertices in the tree. [CZ12]

Definition 1.3. A **directed path** is a sequence of edges that connects a sequence of distinct vertices, where every edge is traversed in a consistent direction. [CZ12]

Definition 1.4. A **node** is a vertex on a tree graph. [CZ12]

2 Background

We define simple graphs by their sets of edges and vertices. An arbitrary graph G is generally defined by (V, E) where V is the number of vertices within the graph and E is the number of its edges.

Putting certain restrictions on a graph can yield interesting results. A tree is a simple graph with two restrictions, as mentioned in definition 1.1 – trees are (1) **connected** and (2) **acyclic**. These two criteria affect both the vertex and edge sets of the graph, changing the values within the graph's fundamental definition of (V, E).

Definition 2.1. A graph is **connected** if for any two vertices x, y within $V(G)$, there is a path whose endpoints are x and y. [CZ12]

Definition 2.2. A **path** of length n from u to v in G is a sequence of n edges $e_1, ..., e_n$ of G for which there exists a sequence $x_0 = u, x_1, ...x_n - 1, x_n = v$ of vertices such that e_i has (for $i = 1, ..., n$) the endpoints $x_i - 1$ and x_i. [CZ12]

Definition 2.3. A graph is described to be **acyclic** if it contains no cycles. [CZ12]

Definition 2.4. A **cycle** is a path where the first and last traversed vertices are the same. [CZ12]

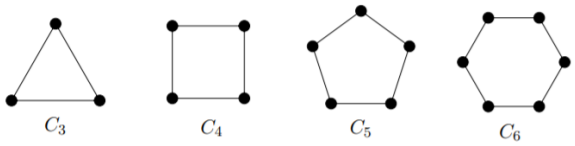
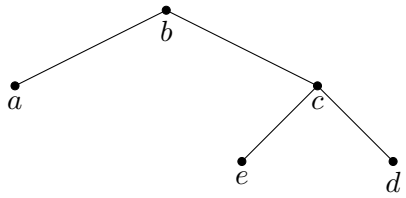


Figure 1: Left to right: cycles with 3, 4, 5, and 6 vertices respectively. A cycle with n vertices is notated C_n.

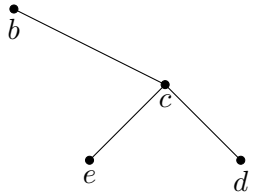
3 Spanning Trees and Bridges



Imagine you're in charge of designing a new network of train lines for the MBTA. You have to connect five (fictional) cities from north to south: Agria, Biloti, Clodo, Durley, and Emmio. You put exactly one path of train tracks connecting every city, so that every city is connected via the same train line. But one day, a tree falls on the train tracks between Biloti and Clodo. Now, there is no way for passengers to get to Durley or any other location further south. For this reason, we call the section of train tracks between Biloti and Clodo a **bridge**.

Definition 3.1. An edge e in a connected graph G is a **bridge** if $G - e$ is disconnected. An edge e in a *disconnected* graph G is a bridge if $G' - e$ is disconnected, where G' is the connected component of G that contains e. [CZ12]

Now let's say that we want to make repairs to our train line system. We only want to focus on the section involving Biloti, Clodo and all cities further south (Durley and Emmio). A graph that only includes these four cities would appear as follows:



We can classify this as a tree subgraph, since it is a smaller graph within our overall train network to which the tree definition still applies. If all vertices were included *and* the subgraph were a tree, we could call our graph a **spanning tree**. The only spanning tree of a tree graph is a copy of the tree graph itself. [FP]

Definition 3.2. A spanning subgraph H of a connected graph G such that H is a tree is called a **spanning tree** of G. [CZ12]

3.1 Kirchhoff's Matrix Tree Theorem

Gustav Kirchhoff, a German physicist and revolutionary electrical engineer, developed a theorem involving matrices to determine the total number of distinct spanning trees in a given graph. [DPW]

Theorem 3.3. *The number of spanning trees in a graph G is given by $det(L_G[i])$ for any i.*

Proof. In this theorem, L refers to the Laplacian matrix, the matrix formed as a result of subtracting the matrix representing the graph's connectedness (adjacency matrix) from the matrix recording the degree of every vertex within the graph (degree matrix).

The adjacency matrix is formed by assigning the same n vertices to both columns and rows, and filling each entry with a value of either 0 or 1, depending on whether or not there is an edge connecting the vertices to which the entry corresponds.

The degree matrix is similarly formed by assigning the same n vertices to columns and rows of the matrix, but its entries look different: all entries, minus those along its diagonal, are equal to 0. Entries along the matrix's diagonal correspond to the same vertex in both column and row, meaning that the entry represents the degree of that exact vertex (and is not undefined, as it would be for an entry combining two vertices).

Recall the formula for expressing the Laplacian matrix based on these two matrices:

$$L = D - A$$

We know that the sum of the nonzero entries in each row and column of D will be equal to that of A. Therefore, when we subtract the two matrices to construct L, we will notice that each row and column of L sums to zero.

If we add any two rows of L together, the sum will be zero. When the sum of two or more matrix rows is equal to zero, the matrix must be **linearly dependent**. All linearly dependent matrices have determinants of 0, so det(L)=0.

To obtain a non-zero determinant, we consider $L[i]$, the matrix formed by deleting the i-th row and column of L. $\det(L[i])$ can be expanded as a sum over permutations. From the Leibnitz formula for the determinant of $L[i]$, a $(n-1)$ by $(n-1)$ matrix:

$$\det(L[i]) = \textstyle\sum_{\sigma \in S_{n-1}} \operatorname{sgn}(\sigma) \prod_{k=1}^{n-1} (L[i])_{k,\sigma(k)}$$

This formula forces each term in the determinant of the Laplacian to select exactly one entry from each row and column of the matrix. Based on the $D - A$ definition of the Laplacian, there are three cases for what this selected entry could be:

$$L_{i,j} = \begin{cases} \deg(i), & i = j, \\ -1, & i \neq j \text{ and } i \text{ is connected to } j, \\ 0, & \text{otherwise.} \end{cases}$$

Based on the Leibnitz formula, we know that if even one of the selected entries is 0, the entire permutation equals 0 and does not contribute meaningful information. Using the definitions of non-zero $L_(i, j)$ cases, we know that all non-zero permutations must be made up of a selection of diagonal $\deg(i)$ entries (distinct points on a graph) and/or off-diagonal -1 entries (edges on a graph).

Within a single permutation, an entry is selected from row k and column σk. Every k has one σk – this means that, graphically, every vertex sends out and receives exactly one directed arrow. A graph made up of these $k \to \sigma k$ permutations is therefore a graph solely made up of vertices with an in-degree and out-degree of 1. From this, we know that the graph must be comprised of disjoint directed cycles covering all vertices.

Permutations that contain any directed cycle of a certain length (at least 2) can be grouped into families whose total contribution to the determinant sum is zero. When summing these cycles to find the determinant of the Laplacian, they will cancel each other out.

After that cancellation, what remains in the determinant sum are selections of matrix entries that do not form any directed cycles involving the remaining vertices. These surviving terms, when interpreted on the graph after removing vertex i, connect all vertices (every k still has one σk) without forming cycles. In other words, the remaining configurations correspond to connected, acyclic edge sets spanning all vertices of the reduced graph.

These configurations are therefore the spanning trees of graph G. Since every configuration contributes exactly once to the determinant sum, $\det L_G$ counts the exact number of spanning trees present in graph G. [DPW]

□

4 Binary Search Trees

For a given graph, there may be many possible spanning trees – just as for a given collection of elements, there may be many possible search trees.

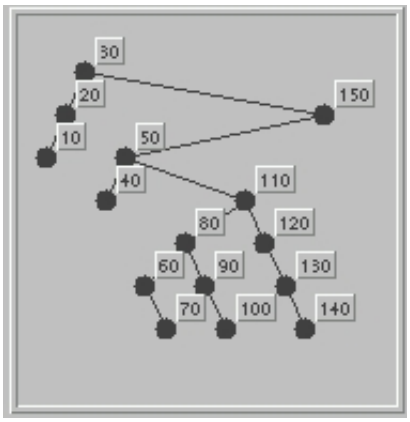


Figure 2: A binary search tree with 15 nodes

A binary tree is defined by the relationship between its elements: each of its nodes must have at most two children. A binary search tree orders and stores data (such as integers). If standard CS operations such as search(x), insert(x) and delete(x) are supported by a binary search tree (where x represents a specific data value), we call such a tree a **dictionary**. [Bie88]

4.1 Red-Black Trees

In a typical binary search tree, all nodes have a specific value and placement on the graph – however, in a red-black tree, one additional piece of information is given for each node. Nodes are assigned colors according to the following restrictions:

1. Every node must either be red or black.
2. Every leaf is black.
3. If a node is red, then the nodes stemming from it are black. Red nodes must not be adjacent.
4. Every path from a node to a leaf, denoted **bh(x)** for the **black-height of a node**, includes the same number of black nodes.
5. The root is black.

These structural rules ensure that red-black trees always remain **balanced**, meaning that their total height never exceeds a logarithmic height (specifically, $2\log_2(n+1)$). The amount of time it takes to search a binary search tree is always proportional to the tree's height, so we know that the time needed for a red-black tree search can be represented as $O\log(n)$. [Mor88]

Red-black trees are most commonly used in computer kernels, such as those of Linux and other operating systems. The trees are used to track multiple processes by execution time and, as a result, allow for efficient scheduling of a machine's tasks. [Lan07]

5 Heaps

Binary heaps are complete binary trees (meaning that every level of the binary tree is filled with the maximum number of nodes possible). There are two types of heap ordering:

Minimum-heap property: the value of each node is $\geq$ the value of its parent; the minimum-value element is at the root.

Maximum-heap property: the value of each node is $\leq$ the value of its parent; the maximum-value element is at the root.

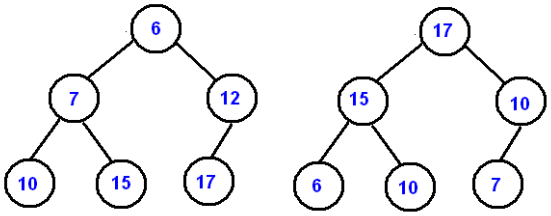


Figure 3: Left to right: minimum heap, maximum heap [Unia]

Heaps are particularly useful when one's objective is to remove the entry with the highest or lowest priority from a data set – for this reason, heaps are often used to implement priority queues. Priority queues manage comparable items based on the maximum heap structure, where items are sorted by importance or urgency (represented by an integer value). Priority heaps are advantageous because the time needed to identify the highest-priority item is constant, no matter the size of the overall data set. [Unia]

5.1 Huffman Trees

Huffman trees also help increase the efficiency of data storage. Named for 20th century computer scientist David A. Huffman, these trees use a process called **variable-length coding**

for 'lossless' data compression. Variable-length coding involves assigning shorter codes to more frequently-occurring data, and longer codes to less frequently-occurring data.

Huffman trees can store multiple different forms of data, but one of their most common uses is storing strings of characters. To build a Huffman tree for a string containing n characters, one must form n initial leaf nodes (containing exactly one letter and its frequency within the string). All of those "partial trees" are then put onto a priority queue according to their frequencies. Then, the two lowest-frequency trees are removed from the queue and joined together to form a larger tree rooted by a node value that is the sum of the two trees' frequencies – this new, larger tree is re-inserted into the priority queue. The process of removing, joining, and re-inserting "partial trees" two at a time is repeated until all trees have been combined into one. The result of this process is a Huffman tree! [Sul]

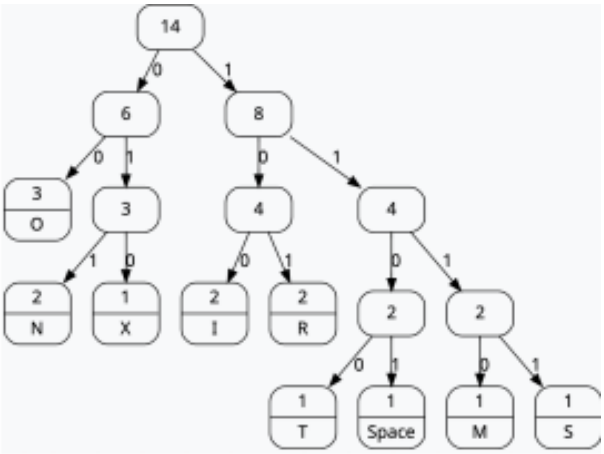


Figure 4: A Huffman tree storing the string "TONI MORRISON." The character "X" is included to indicate the end of the string. [Unib]

Huffman coding is most efficient at saving storage space for strings when there is a large variation in letters' frequencies. Coding a typical text file with the Huffman technique can save roughly 40 percent of file space compared to ASCII or other traditional methods of coding. [Unib]

6 Prefix Trees

Prefix trees, or "tries," are another type of tree structure used for efficiency in managing a string-based dataset.

If one wants to search through a list of N words made up of L letters each, this searching process would ordinarily take $O(N * L)$ time. However, using a trie, the amount of time needed for the search is reduced to $O(L)$, which is independent of N, the number of words in the list. This makes tries incredibly valuable for optimizing the time it takes to search through large datasets, where the large value N would typically create a significant increase in necessary search time. [MR]

Tries are constructed using a null (no character) root from which all data strings stem. Strings containing the same prefix reuse the same path in the tree, and the * symbol (or another distinguishing mark, such as ') signifies the end of a string. For example, the words "app," "ape" and "apple" would appear on a trie in the following way:

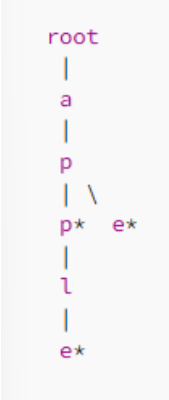


Figure 5: A trie storing the strings "app," "ape," and "apple." [Gup26]

Searching through a prefix tree determines whether *any* included word starts with a given prefix. This makes tries extremely applicable to search engines and autofill – imagine the large database of suggested search queries as one very large prefix tree. A user types in just a few letters and the search engine searches that trie for the user's particular prefix, providing them with data entries in the trie (suggested searches) that contain the user's input.

In addition to search engines' autocomplete function, tries are also used within spell checkers, word games (like make-your-own-crosswords) and dictionaries. [Gup26]

7 Decision Trees

A decision tree is a specific type of binary tree consisting of two types of nodes: decision nodes and leaf nodes. The purpose of decision nodes is to split the dataset recursively until only leaf nodes remain. Every decision node has exactly two edges that are assigned boolean values, splitting the dataset based on whether or not data entries satisfy the decision node's criteria. [Dut21]

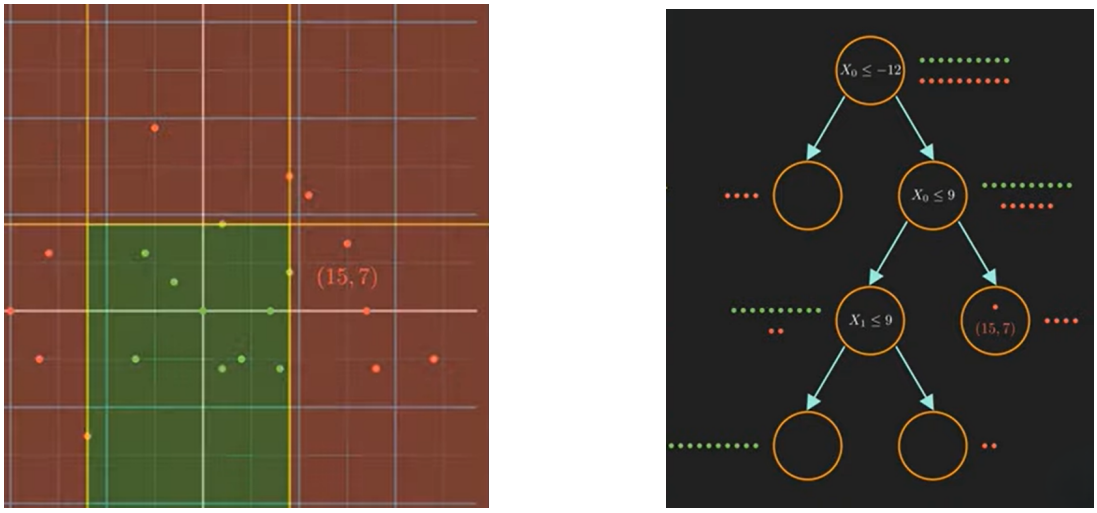


Figure 6: A decision tree sorting red and green points on a Cartesian graph (right), creating corresponding colored graph sections (left) once the process is complete. [Dut21]

Decision trees are valued for their "white-box" nature, meaning that the reasoning behind all tree outcomes is transparent and can be traced from the tree's leaves all the way back to its root. These trees are often found in a "flowchart" form in everyday life, but are also commonly used

in artificial intelligence (to aid in natural language processing) and cybersecurity (to analyze network traffic patterns and prevent attacks). [Sub25]

8 Acknowledgements

The author is truly grateful for mentor Nathra Ramrajvel, graph theory classmate Eileen Lee, program coordinators Mary Stelow and Paige Bright, as well as everyone else on the PRIMES Circle team! Participating in PRIMES Circle has been a highlight of my year and an experience that won't soon be forgotten.

9 References

References

[Bie88] Henning Biermann. Binary search trees. https://cs.nyu.edu/algvis/java/bst.html, 1988.

[Chu10] Fan Chung. Graph theory in the information age. *Notices of the AMS*, 2010.

[CZ12] G. Chartrand and P. Zhang. *A First Course in Graph Theory*. Dover books on mathematics. Dover Publications, 2012.

[DPW] Sloan Nietert David P. Williamson. Lecture 8. Lecture notes on Kirchhoff's Matrix-Tree Theorem from Cornell University's ORIE 6334 class.

[Dut21] Sujan Dutta. Decision tree classification clearly explained. https://www.youtube.com/watch?v=ZVR2Way4nwQ, 2021.

[FP] Iliano Cervesato Frank Pfenning. Lecture 25: Spanning trees. Lecture notes from Carnegie Mellon University's 15-122 class.

[Gup26] Eshita Gupta. Trie: The data structure that understands prefixes. *Medium*, 2026.

[Lan07] Rob Landley. Red-black trees (rbtree) in linux. 2007. Listed on website "The Linux Kernel".

[Mor88] John Morris. Red-black trees. https://www.eecs.umich.edu/courses/eecs380/ALG/red_black.html, 1988.

[MR] David Kempe Mark Redekopp. Tries. Slides for University of Southern California's CSCI 104 class.

[Mul88] Henry Martyn Mulder. To see the history for the trees. https://www.researchgate.net/publication/249328601_To_see_the_history_for_the_trees, 1988.

[Sub25] Abinaya Subramaniam. Branching out with decision trees: Understanding splits, nodes, and leaves. *Medium*, 2025.

[Sul] David G. Sullivan. Binary trees and huffman encoding. Slides for Harvard University class "Computer Science E-22.".

[Tos15] Besjana Tosuni. Graph theory in computer science - an overview. *International Journal of Academic Research and Reflection*, 2015.

[Unia] Carnegie Mellon University. Binary heaps. Published on www.andrew.cmu.edu website, no author listed.

[Unib] Pennsylvania State University. Huffman coding trees. Slides for class CIT 5940, no author listed.