Trees

Table of Contents

[Minimally Connected Graphs 4](#_Toc141041231)

[Pendant Vertices 5](#_Toc141041232)

[Subsequences 6](#_Toc141041233)

[Distances and Centers 7](#_Toc141041234)

[Rooted Trees 9](#_Toc141041235)

[Binary Trees 10](#_Toc141041236)

[Pendant and Internal Vertices 10](#_Toc141041237)

[Level of a Vertex 11](#_Toc141041238)

[Counting Trees 14](#_Toc141041239)

[Spanning Trees 15](#_Toc141041240)

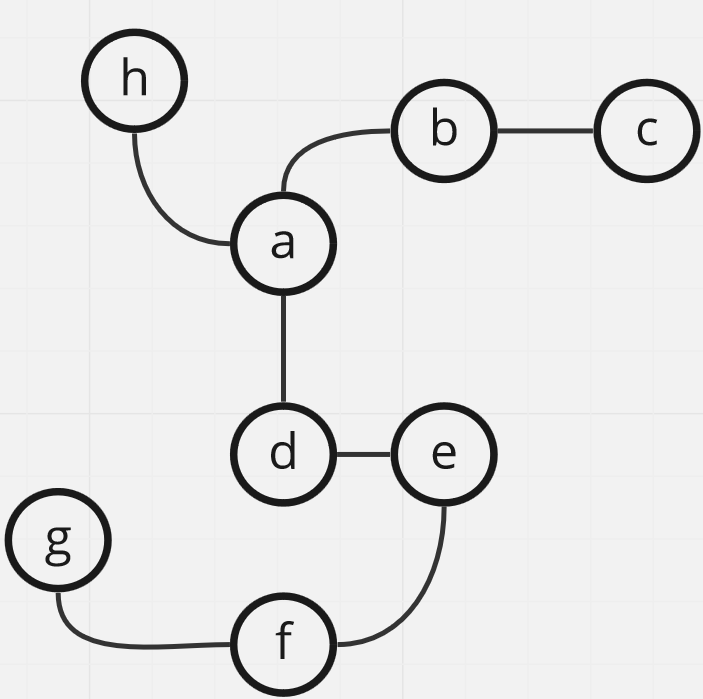
[Fundamental Circuits 19](#_Toc141041241)

[Finding All Spanning Trees 20](#_Toc141041242)

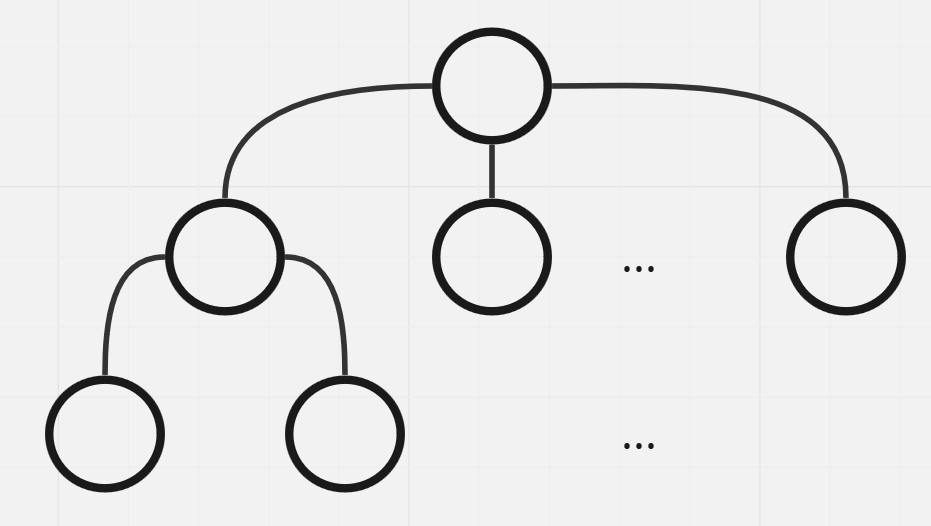
[Tree Graph 21](#_Toc141041243)

[Spanning Tree in a Weighted Graph 21](#_Toc141041244)

A **tree** is a **connected graph** with **no circuits**. As such, it is also a **simple graph** with no self-loops or parallel edges.



Trees are useful in many practical applications. Suppose we have a post office that must divide letters into paths based on the zip code of the delivery address. We can organize this system by having one post office send letters to a certain area (branch) based on the first digit of the zip code, then another post office repeat the process based on the second digit and so on. This creates a **decision tree** or **sorting tree** as shown below.



Theorem 3.1 There is one and only one path between every pair of vertices in a tree.

Proof:

Suppose there are 2 paths between two vertices and . In this case, we can use one path to go from to and another to go from to . This creates a circuit, and by definition, trees cannot have circuits. Thus, this situation is not possible for a tree.

Theorem 3.2: If a graph has one and only one path between every pair of vertices, then is a tree.

Proof:

Since a path exists between all the vertices, the graph is **connected**. Since there is only one path, there cannot be a **circuit**. A connected graph with no circuits is a tree.

Theorem 3.3: A tree with vertices has edges.

Proof:



Let the theorem be true for a tree with less than vertices. We can see in the diagrams above that it is true for and .

For a tree with edges, let there be an edge between two vertices and , where and are connected by only this path. Removing this edge thus results in the creation of two components with vertices and vertices respectively, and . We can see that and .

If , the above proof shows that there must be and edges in each of the two components, which gives a total of edges. If we add back the edge we removed, there are edges in the original tree. Thus, the theorem is also true for . Recursively continuing this process for larger values of proves that the theorem is true for any arbitrary value of .

Theorem 3.4: Any connected graph with vertices and edges is a tree.

## Minimally Connected Graphs

A graph where removing any edge causes it to become disconnected is called a **minimally connected graph**.

Theorem 3.5: A graph is a tree if and only if it is minimally connected.

Theorem 3.6: A graph with vertices and edges and no circuits is connected.

Proof: Suppose has vertices, edges and no circuits but is disconnected. If we pick two vertices and from the two disconnected components and connect them via an edge, we have edges in total but the graph still does not have a circuit. A graph that is connected but has no circuits forms a tree, but a tree must have edges. Thus, this case is not possible.

Similar to the above, we also have a few more scenarios:

1. A graph that is connected and circuitless is a tree.
2. A graph that is connected and has edges is a tree.
3. A graph that is circuitless and has edges is a tree.
4. A graph with a path between every pair of vertices is a tree.
5. A graph that is minimally connected is a tree.

## Pendant Vertices

For a tree with vertices and edges, there should be a total degree of .

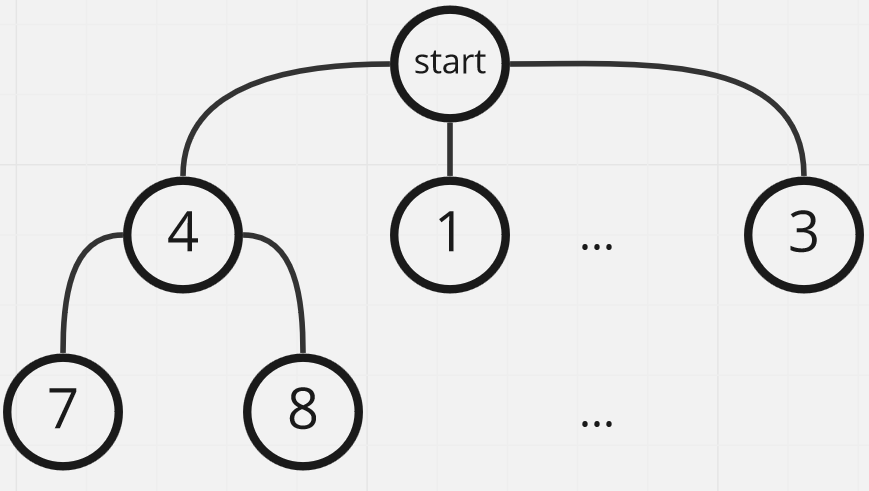
Suppose , meaning there are vertices, edges and a total degree of . On average, every vertex should have a degree of . This means every vertex cannot have a degree of . There are also no vertices with a degree of , since a tree is connected. This means that there are **pendant vertices** in a tree. Additionally, it also means that there are at least 2 pendant vertices, since the number of vertices with odd degree must be even.

Theorem 3.7: In a tree with vertices, there are at least 2 pendant vertices.

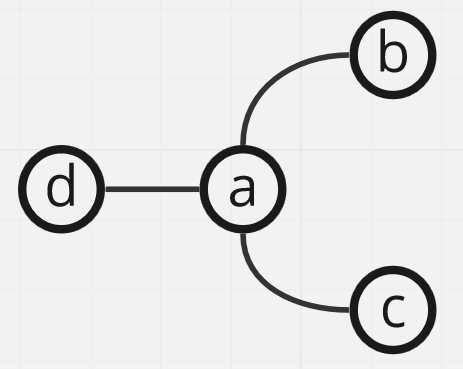
## Subsequences

A **subsequence** of a sequence is a subset that maintains the order of the sequence while removing 0 or more elements from the sequence. For example, in the sequence 4, 1, 13, 7, 0, 2, 8, 11, 3, possible subsequences include 1, 2, 8, 11 or 4, 7, 8, 11 or 1, 7, 8, 11.

To find the **largest monotonically increasing subsequence** of a sequence, we can construct a tree. First, we create a staring node to which every value is connected as a vertex. To each vertex, we connect the values that are larger. The largest branch formed is the largest monotonically connected subsequence.



## Distances and Centers



Intuitively, we know that the center of the graph above is . This is easy to find for small graphs like this one, but it becomes difficult for complicated graphs.

Formally, the center of a graph is the vertex which has the **least overall distance** to the other vertices.

For a connected graph, the **distance** between two vertices, , is the length of the **shortest path** between them.

There are a few rules to calculating the distance:

1. The distance must be **non-negative** and can be 0 if and only if .
2. The distance must be **symmetric**, i.e., .
3. The distance must maintain the **triangle inequality**, .

A function which satisfies all of these rules is called a **metric**.

The shortest distance between two vertices meets all of these criteria.

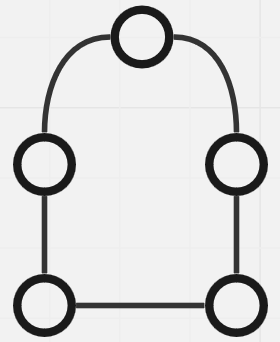
1. Since the length of the path is determined by the number of edges, it cannot be negative. The only way it can be 0 is if there are no edges between the vertices, i.e., .
2. Since the graph is undirected, symmetry is maintained.
3. The triangle inequality is maintained since if was a smaller value than , then that would be the shortest path.

Theorem 3.8: The distance between two vertices is a metric.

The maximum distance between one vertex and any other vertex in the graph is called the **eccentricity** of the vertex, .

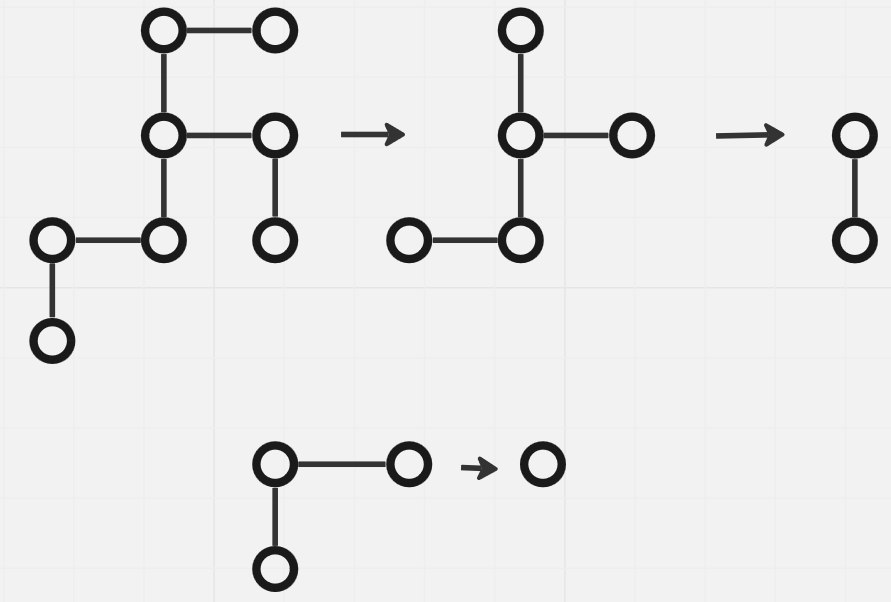
The **centre** of a graph is thus the vertex which has the **minimum eccentricity**. The eccentricity of the centre is called the **radius** of the graph. The maximum eccentricity in the graph is called the **diameter**.

It is possible for a graph to have multiple centres, but not a tree.



Konig’s Theorem: The number of centres in a tree is either one or two.

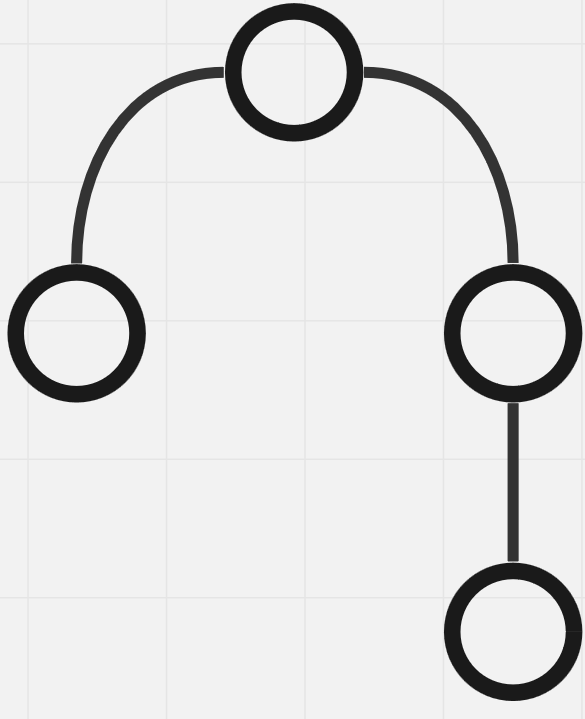
Proof: Suppose we have a tree from which we are repeatedly removing pendent vertices, as shown below.



At each removal, for every vertex remaining decreases by one. This means that the order of eccentricities remains the same, meaning the centre remains the same. As shown above, we can remove pendants repeatedly until we have either one or two vertices remaining. This means there can only be one or two centers.

## Rooted Trees

A **rooted tree** is a tree where one of the vertices has been distinguished and labeled as the root.

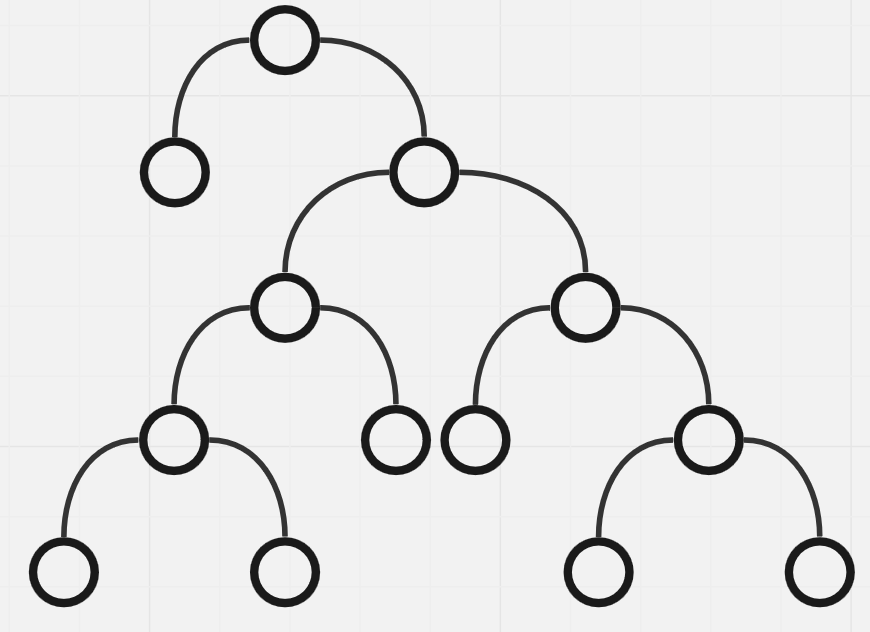


We should assume that a tree is non-rooted unless otherwise specified.

Rooted trees have a variety of applications which we will be exploring.

## Binary Trees

A **binary tree** is a tree with exactly one vertex with a degree of two and every other vertex with a degree of one or three.



For a tree with vertices, this means that there are vertices with an odd degree. According to theorem 1.1, must be even. This means must be odd.

### Pendant and Internal Vertices

Let the number of **pendant vertices** be .

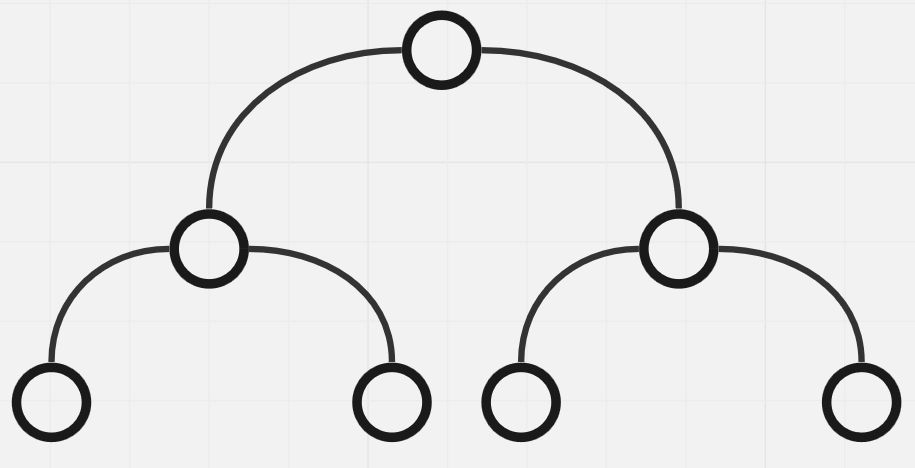
In a binary tree, the non-pendant vertices are called **internal vertices**. The number of internal vertices is given by . Thus, the number of pendant vertices is one more than the number of internal vertices.

### Level of a Vertex

The **level** of a vertex, , is the distance of the vertex from the root. The level of the root is .

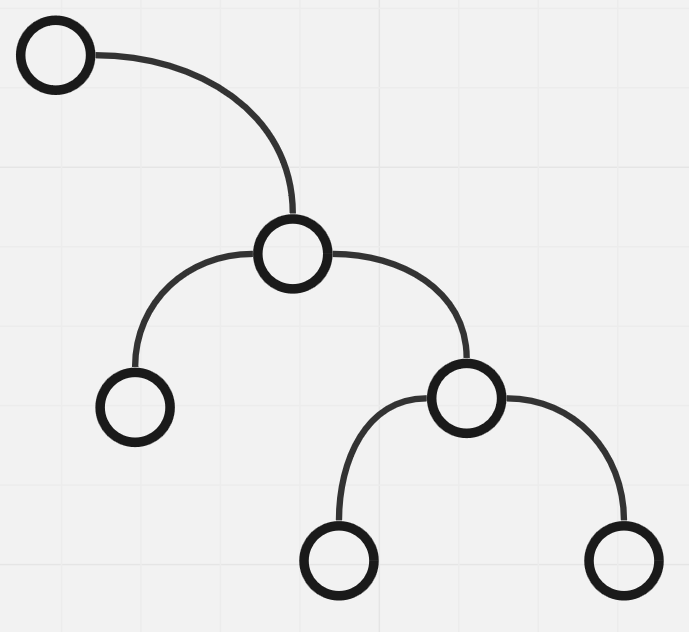
The further the pendant vertices are, the longer an algorithm will take to work with the tree. Thus, the height of the tree, which is the **maximum level**, , is an important value.

Suppose we have a -level binary tree. The minimum value of occurs when all of the vertices are packed.

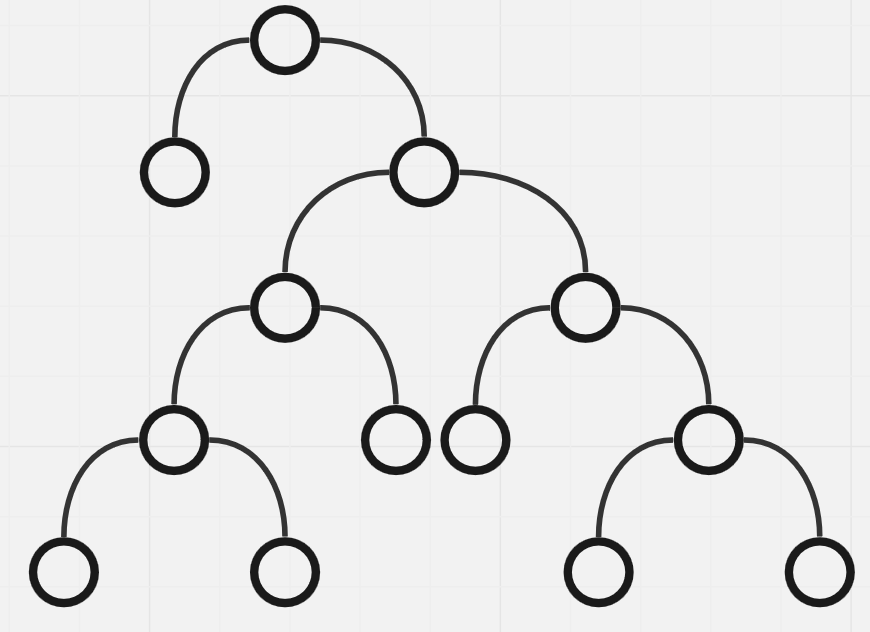


The total number of vertices is . Since this is minimum value of , the inequality is actually since this is the maximum number of vertices that will fit in levels. Thus, .

We can also have a situation where the height is the minimum possible one but the vertices are not fully packed. This occurs for , where . Thus, .

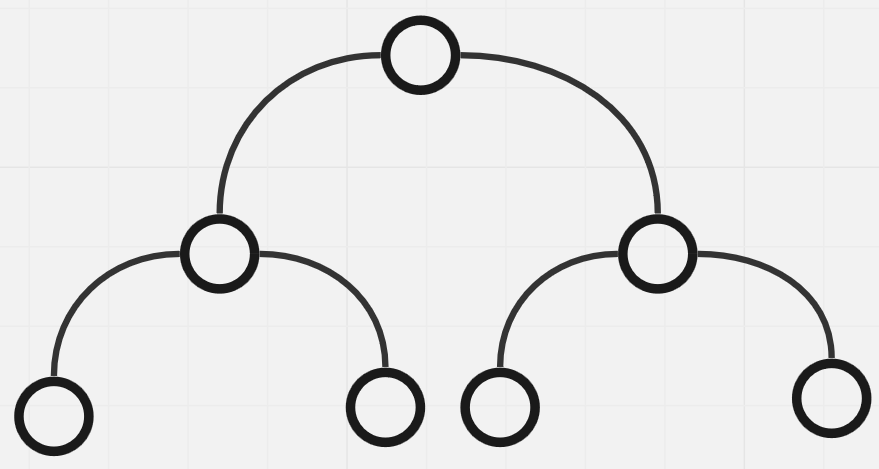


The above shape will result in the maximum possible height while still being a binary tree. There are 2 nodes per level, so .

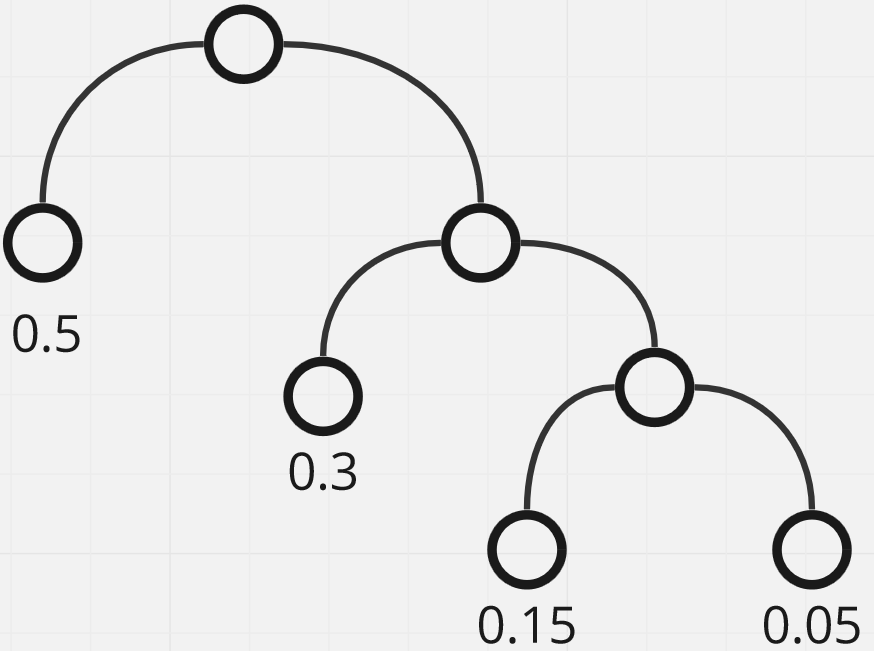


The sum of the levels of the pendant vertices is called the **path length**. For the diagram above, the path length is . The lower the path length, the faster an algorithm will be able to work with the tree. The minimum path length occurs are the minimum value of .

The concept of a **weighted path length** is also useful in practical applications, where the levels of each pendant vertex also have a weight. Suppose we have a vending machine that accepts coins of 0.5p, 1p, 2p and 5p. The machine needs to determine which type of coin has been inserted.



Normally, the decision tree for this would be like the one shown above. To make any decisions, there will always be 2 experiments. However, what if we also know that the probabilities of each type of coin being inserted are 0.5, 0.3, 0.15 and 0.05 respectively. This information allows us to use the probabilities as weights. In this case, the tree with the minimum weighted path length will have the vertices that have a higher probability of occurring at the top, as shown below.

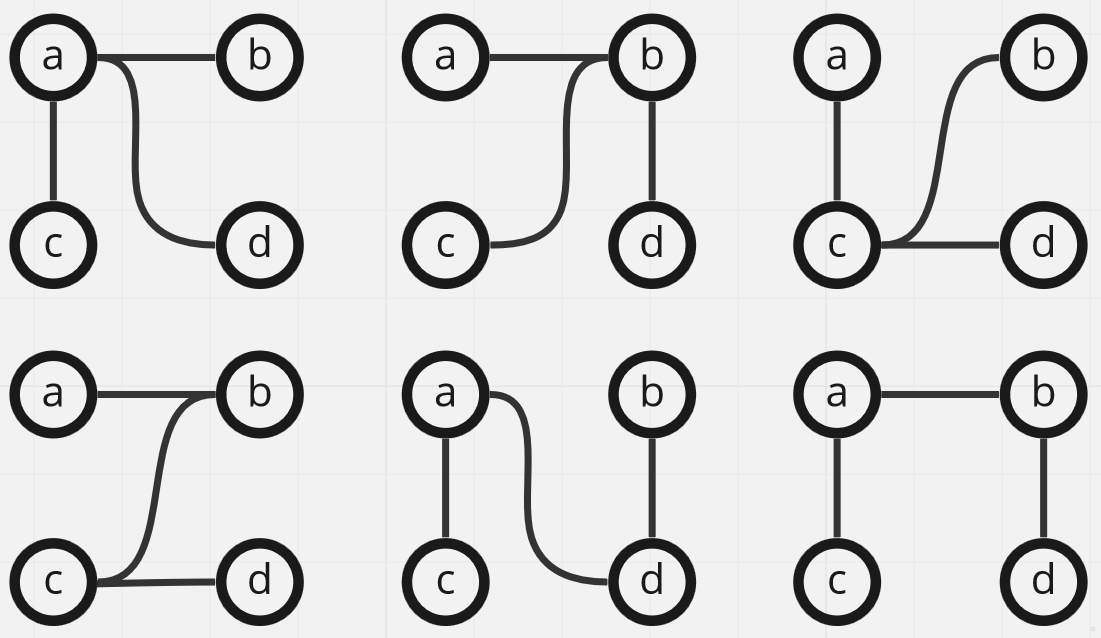


This tree minimizes the number of experiments the machine needs to perform in the long term, 1.7 on average.

## Counting Trees

**Counting Trees** were originally used to calculate the number of structured isomers a paraffin can have. Paraffins have the chemical composition of . If each and each is a vertex, there are nodes and edges. Since there are vertices and edges, this forms a tree. The problem is to find the number of ways in which we can represent a tree.

Suppose . Possible setups for the tree are shown below.



There are a total of 16 possible trees for . As can be seen, some of them are actually isomers and are only being differentiated based on the label.

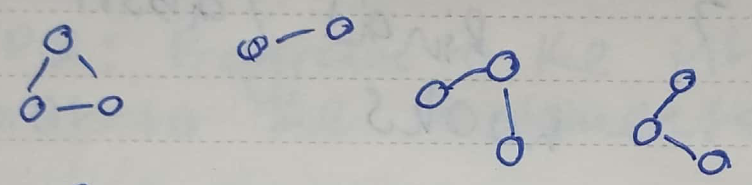
Theorem 3.10: The number of labeled trees with vertices, where , is .

The proof for this theorem is quite complicated and is thus not presented here.

However, the original problem of finding the number of structured isomers does not use labels. It was noticed that the nodes are always pendants and do not contribute to the number of structures. This means we only need to calculate the number of ways the nodes can be connected. This was also verified by the fact that results in exactly possible ways for the structures to be formed, and Butane (with 4 Carbons) has exactly 2 isomers.

## Spanning Trees

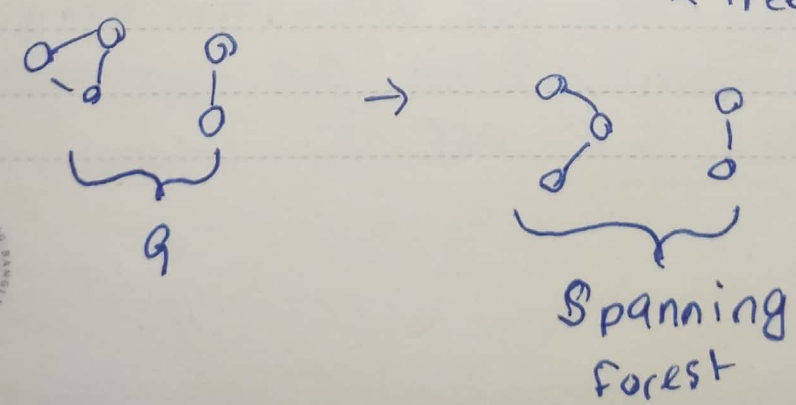
The **number of subgraphs** that can be generated from a single graph with edges is . This is because for each edge, we have two options, either keeping it or removing it.



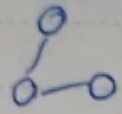
Some of the subgraphs created from a graph are **trees**. If the subgraph also happens to be an **induced subgraph**, i.e., it has all of the vertices of the original graph, the subgraph is called a **spanning tree**, .

Due to the way spanning trees are defined, they are the **largest trees** that can be created from the graph. Since there are no more edges to add, we cannot create a larger tree. As such, spanning trees are also called **maximum tree subgraphs**.

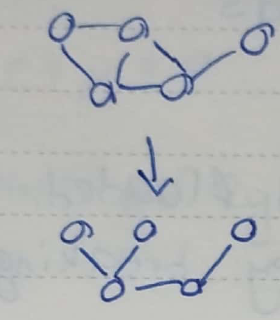
If a graph is **disconnected**, it is not possible to create a spanning tree from it. However, each component of the graph can be used to create a spanning tree as though they were an individual graph. For components, we will thus have spanning trees. This setup is called a **spanning forest**.



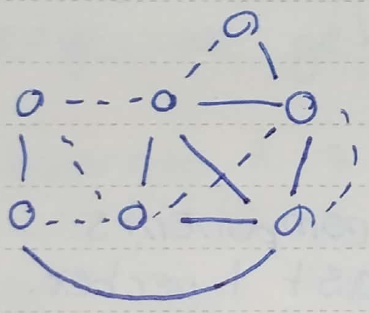
If a **connected graph** has **no circuits**, the graph itself is a spanning tree.



Otherwise, we need to create a subgraph. We do this by **identifying circuits** in the graph one by one and **removing edges** form the circuits until none remain. The resulting tree will be a spanning tree.



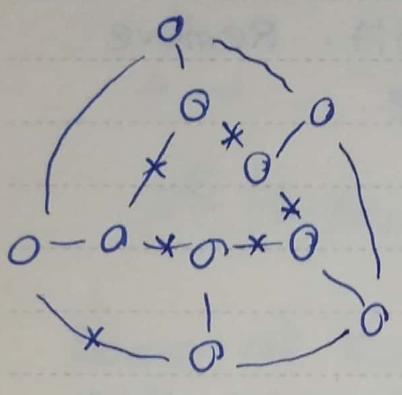
Theorem 3.11: Every connected graph has at least one spanning tree.



In the diagram above, the solid edges are not part of the spanning tree and the dotted edges are. The edges that are part of the spanning tree are called the **branches** while those that are not, are called the **chords**. Thus, . Here, both and contain all of the vertices of the graph. contains all of the branches, while contains all of the chords. The set of chords is called the **chord set**.

Notice that which edges are branches and which edges are chords depends entirely on how we decide to create the spanning tree. If we choose to remove different edges from the circuits, we will end up with a different spanning tree with different branches and different chords.

Theorem 3.12: With respect to a spanning tree, a connected graph with vertices and edges has branches and chords.



Consider that the graph above represents a farmland that has been completely flooded. The edges represent walls here. We need to find the **minimum number of edges** that need to be broken for the farmland to be unflooded. The crossed edges show one possible set of walls that can be broken to achieve this. Again, we are just removing edges from circuits until there are none left, meaning we are creating a spanning tree. The edges we remove are the chords, so the minimum number of walls to break is the number of chords in the spanning tree.

In the graph above, we originally had vertices and edges. We broke edges.

Suppose a graph has components. To even be a component, there needs to be at least one vertex in the component. Thus, we know that .

If the th component has vertices, the minimum number of edges is . Thus, . For all of the components, . and there are components, so .

The two values we have just found can be re-written as and . is called the **rank**, , of the graph. is called the **nullity**, , of the graph. Thus, and .

For a spanning tree, , the number of branches, and , the number of chords. Thus, .

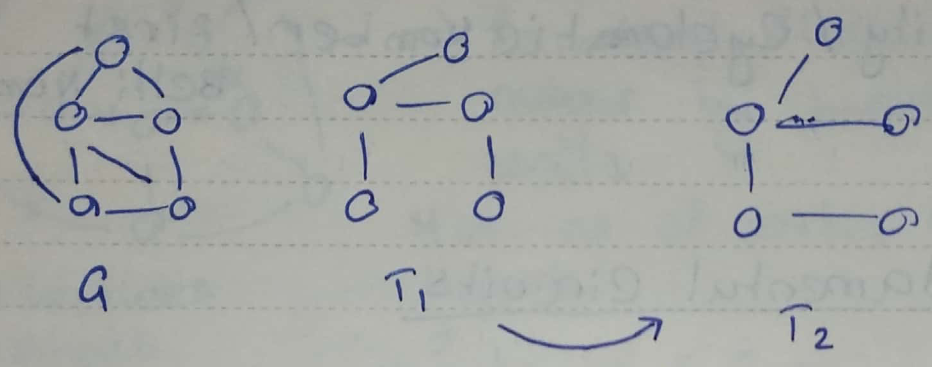
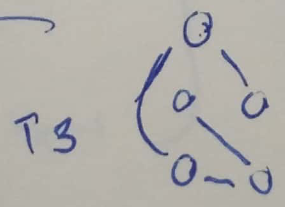
### Fundamental Circuits

If we have an existing tree and add any edge to it, we end up with a circuit in the tree.

Theorem 3.13: A connected graph is a tree if and only if adding an edge between adding an edge between any two vertices in creates a circuit.

For a spanning tree, the circuit that we create is created by adding a chord. The circuit itself is called a **fundamental circuit**. Since there are chords, there are possible fundamental circuits. Again, since the chord set can change depending on how we create the spanning tree, the fundamental circuits also change.

### Finding All Spanning Trees

The diagrams above show a few possible spanning trees for the graph . Suppose we have one spanning tree, . To get to the another spanning tree, we add a chord, thus creating a circuit, and remove a different edge from the circuit. This creates a new spanning tree. By repeating this process, we can create all of the spanning trees. This process is called **cycling interchange**.

The number of edges that are present in one spanning tree but not in another spanning tree is called the **distance**, , of those spanning trees. For the spanning trees shown above, while .

An alternative way of defining the distance is in reference to the **ring sum**. The ring sum of two spanning trees , is the number of uncommon edges between the trees. The distance is half of this value.

Theorem 3.14: The distance of between two spanning trees is a metric.

Proof: Too complicated. Skip.

Theorem: 3.15: Starting from any spanning tree, we can obtain every other spanning tree by performing successive cyclic interchanges.

Proof: Too complicated. Skip.

The **maximum distance** between two spanning trees is equal to the maximum number of cyclic interchanges. This occurs when all of the edges from the original spanning tree have been replaced, meaning the value is . Thus, . Another way of looking at this is that all of the branches are removed and all the chords have been added, meaning . It is not guaranteed that we will be able to remove all of the branches or be able to add all of the chords, so the maximum value is .

### Tree Graph

If we consider each spanning tree that can be created from a single graph as a node and connect the nodes that have a distance of 1, we end up with a graph. This graph is called a **tree graph**. The centre of the graph is called a **central tree**.

### Spanning Tree in a Weighted Graph

Consider that we are tasked with constructing roads between cities with the condition that all of the cities must be connected. The cost of constructing roads between different pairs of cities varies and we must find the minimum cost with which the project can be completed.

Essentially, the above requirement is to create a spanning tree with the minimum cost, called a **minimal spanning tree**. It can be created using one of two algorithms, Kruskal’s algorithm and Prim’s algorithm.

In **Kruskal’s algorithm**, we first sort the edges based on their cost. Next, we selected edges one by one in an ascending manner such that the edges do not create circuits. We will end up with exactly edges that create a minimal spanning tree.

In **Prim’s algorithm**, we work with the vertices instead. We select any vertex randomly and then create a set of edges that are incident from that vertex. Next, we select the edge from this set which has the minimum cost. This edge takes us to another vertex, so we now have two vertices. We recreate the set of edges using the edges incident with the two vertices and repeat the process. This continues until we have edges, which gives us a minimal spanning tee.

In addition to the problem above, we can add another constraint for the **maximum degree** any one of the nodes has. If we set the maximum degree to 2, then the resulting spanning tree will have exactly one edge entering each node and one edge leaving it. Thus, we have created a **Hamiltonian Path**.