Induction & Recursion

1. **Sequences** is a special type of function in which the domain is a set of consecutive integers. A geometric sequence is a sequence of real numbers where each term after the initial term is found by taking the previous term and multiplying by a fixed number called the common ratio. A geometric sequence can be finite or infinite.

2. **Recurrence relations** Some sequences are most naturally defined by specifying one or more initial terms and then giving a rule for determining subsequent terms from earlier terms in the sequence. A rule that defines a term an as a function of previous terms in the sequence is called a recurrence relation.

3. **Summations**

   -

4. **Mathematical induction** The **base case** establishes that the theorem is true for the first value in the sequence. The **inductive step** establishes that if the theorem is true for k, then the theorem also holds for k + 1

   -

5. **Strong induction and well-ordering** The principle of strong induction assumes that the fact to be proven holds for all values less than or equal to k and proves that the fact holds for k+1. By contrast the standard form of induction only assumes that the fact holds for k in proving that it holds for k+1.

6. **Loop invariants** The field of **program verification** is concerned with formally proving that programs perform correctly. A program's correct behavior is defined by stating that if a **pre-condition** is true before the program starts, then the program will end after a finite number of steps and a **post-condition** is true after the program ends.

7. **Recursive definitions** In a **recursive definition** of a function, the value of the function is defined in terms of the output value of the function on smaller input values. One can use the recursive definition for n! to determine the value of the factorial function on a particular value for n by starting at 0, multiplying 0! by 1 to get the value of 1! then multiplying by 2 to get 2!, and so on, until the desired n! has been reached. The process is called recursion. **Recursion** is the process of computing the value of a function using the result of the function on smaller input values.

8. **Structural induction** is a type of induction used to prove theorems about recursively defined sets that follows the structure of the recursive definition.

9. **Recursive algorithms**

   -

10. **Induction and recursive algorithms**

    -

11. **Analyzing the time complexity of recursive algorithms**

-

12. **Divide-and-conquer algorithms: Introduction and mergesort**

-

13. **Divide-and-conquer algorithms: Binary search**

-

14. **Solving linear homogeneous recurrence relations**

-

15. **Solving linear non-homogeneous recurrence relations**

-

16. **Divide-and-conquer recurrence relations**

-

Integer Properties

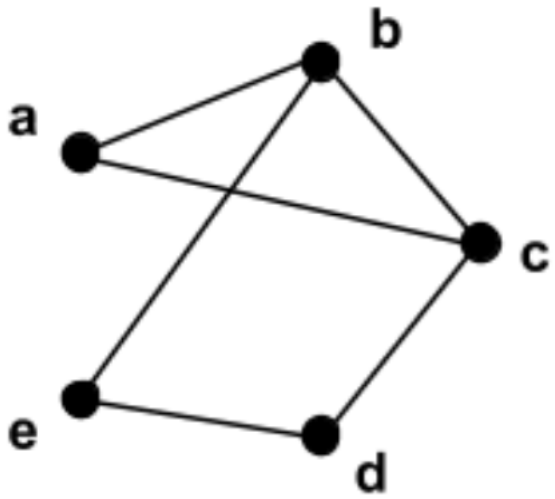
1. **The Division Algorithm**
   - **Integer division**
2. **Modular arithmetic**
   - **Modular arithmetic**
3. **Prime factorizations**
   - **prime**
   - **composite**
   - **prime factorization**
   - **non-decreasing**
   - **multiplicity**
4. **Factoring and primality testing**
5. **Greatest common divisor and Euclid's algorithm**
   - **GCD Theorem**
   - **Euclid's algorithm for finding the greatest common divisor**
   - **The extended Euclidean algorithm**
6. **Number representation**
   - A digit in binary notation is called a **bit**. Numbers represented in **base** b require b distinct symbols and each place value is a power of b.
7. **Fast exponentiation**
   - Computing a power of x by repeated multiplication by x
   - Computing a power of x by repeated squaring
8. **The RSA cryptosystem**
   - In **public key cryptography**, Bob has an **encryption key** that he provides publicly so that anyone can use it to send him an encrypted message. Bob holds a matching **decryption key** that he keeps privately to decrypt messages. While anyone can use the public key to encrypt a message, the security of the scheme depends on the fact that it is difficult to decrypt the message without having the matching private decryption key.
   - Encryption
   - Decryption

Intro to Counting

1. **Sum and product rules**
    - **Product Rule**
    - **Sum Rule**
2. **The bijection rule**
    - **Bijection Rule**
3. **The generalized product rule**
    - **Generalized Product Rule**
4. **Counting permutations**
    - **Permutation**
5. **Counting subsets**
    - **Subset**
    - **r-combination**
6. **Counting by complement**
    - **Counting by complement**
7. **Permutations with repetitions**
    - **Permutation with repetition**
8. **Counting multisets**
    - **Multisets**
9. **Inclusion-exclusion principle**
    - **Principle of inclusion-exclusion**

Graphs

1. **Introduction to graphs** Directed graphs were introduced in the context of relations. Here we are concerned with **undirected graphs**. In an undirected graph, the edges are unordered pairs of vertices, which is useful for modeling relationships that are symmetric. A graph consists of a pair of sets (V, E), where V is a set of vertices and E is a set of edges. A graph is **finite** if the vertex set is finite. This material will only be concerned with finite graphs. A single element of V is called a **vertex** and is usually represented pictorially by a dot with a label. Each edge in E is a set of two vertices from V and is drawn as a line connecting the two vertices. In the graph below, the vertex set is V = {a, b, c, d, e}. The graph has six edges. **Parallel edges** are multiple edges between the same pair of vertices. Imagine a graph whose vertex set is a set of cities and whose edges are roads connecting pairs of cities. It is possible for there to be two different roads between the same two cities. In defining graphs with parallel edges, it would be important to have an additional label besides the two endpoints to specify an edge in order to distinguish between different parallel edges. A graph can also have a **self-loop** which is an edge between a vertex and itself. The graph below has two parallel edges between vertices a and b. There is also a self-loop at vertex c.



- If there is an edge between two vertices, they are said to be **adjacent**. In the graph above, d and e are adjacent, but d and b are not adjacent.
- Vertices b and e are the **endpoints** of edge {b, e}. The edge {b, e} is **incident** to vertices b and e.
- A vertex c is a **neighbor** of vertex b if and only if {b, c} is an edge. In the graph above, the neighbors of b are the vertices a, c, and e.
- In a simple graph, the **degree** of a vertex is the number of neighbors it has. In the graph above, the degree of b is 3 and the degree of vertex a is 2. The degree of vertex b is denoted by deg (b).
- The **total degree** of a graph is the sum of the degrees of all of the vertices. The total degree of the graph above is $2 + 3 + 3 + 2 + 2 = 12$.

2. **Graph representations**

   - In the **adjacency list** representation of a graph, each vertex has a list of all its neighbors. Note that since the graph is undirected if vertex a is in b's list of neighbors, then b must also be in a's list of neighbors.
   - The **matrix** representation for a graph with n vertices is an n by n matrix whose entries are all either 0 or 1, indicating whether or not each edge is present. If the matrix is labeled M, then $M_{i,j}$ denotes the entry in row i and column j. For a matrix representation, the vertices of the graph are labeled with integers in the range from 1 to n. Entry $M_{i,j=1}$ if and only if {i, j} is an edge in the graph. Since the graph is undirected, $M_{i,j} = M_{j,i}$ because {i, j} and {j, i} refer to the same edge which is either present in the graph or not. Thus the matrix representation of an undirected graph is symmetric about the diagonal, meaning that it is a mirror image of itself along the diagonal extending from the upper left corner to the lower right corner.

3. **Graph isomorphism**: Two graphs are said to be **isomorphic** if there is a correspondence between the vertex sets of each graph such that there is an edge between two vertices of one graph if and only if there is an edge between the corresponding vertices of the second graph. The graphs are not identical but the vertices can be relabeled so that they are identical.

   - Let G = (V, E) and G' = (V',E'). G and G' are isomorphic if there is a bijection f: V $\rightarrow$ V' such that for every pair of vertices x, y $\in$ V, {x, y} $\in$ E if and only if $\{f(x), f(y)\} \in$ E'. The function f is called an **isomorphism** from G to G'.

4. **Walks, trails, circuits, paths, and cycles**:

   - A **walk** from v_0 to v_l in an undirected graph G is a sequence of alternating vertices and edges that starts and ends with a vertex:
     $\langle v_0, \{v_0, v_1\}, v_1, \{v_1, v_2\}, v_1, \ldots, v_{l-1}, \{v_{l-1}, v_l\}, v_l \rangle$
   - The vertices just before and after each edge are the two endpoints of that edge.
   - Since the edges in a walk are completely determined by the vertices, a walk can also be denoted by the sequence of vertices: $\langle v_0, v_1, \ldots, v_l \rangle$
   - The sequence of vertices is a walk only if $\{v_{i-1}, v_i\} \in$ E for each i = 1, 2,...,l. Two consecutive vertices ..., v_{i-1}, v_i, ... in a walk represent an occurrence of the edge $\{v_{i-1}, v_i\}$ in the walk.
   - The **length** of a walk is l, the number of edges in the walk.
   - An **open walk** is a walk in which the first and last vertices are not the same. A **closed walk** is a walk in which the first and last vertices are the same.
   - A **trail** is an open walk in which no edge occurs more than once.
   - A **circuit** is a closed walk in which no edge occurs more than once.
   - A **path** is a trail in which no vertex occurs more than once.
   - A **cycle** is a circuit of length at least 1 in which no vertex occurs more than once, except the first and last vertices which are the same.

5. **Graph connectivity**: In an undirected graph, if there is a path from vertex v to vertex w, then there is also a path from w to v. The two vertices, v and w, are said to be **connected**. A vertex is always considered to be connected to itself. If the graph represents a road or communication network, then it is very desirable for every pair of vertices to be connected. The property of being connected can be extended to sets of vertices and the entire graph:

    - A set of vertices in a graph is said to be connected if every pair of vertices in the set is connected.
    - A graph is said to be connected if every pair of vertices in the graph is connected, and is **disconnected** otherwise.
    - A **connected component** is a maximal set of vertices that is connected. The word "maximal" means that if any vertex is added to a connected component, then the set of vertices will no longer be connected.
    - A vertex that is not connected with any other vertex is called an **isolated vertex** and is therefore a connected component with only one vertex.
    - An undirected graph G is **k-vertex-connected** if the graph contains at least k + 1 vertices and remains connected after any k - 1 vertices are removed from the graph. The **vertex connectivity** of a graph is the largest k such that the graph is k-vertex-connected. The vertex connectivity of a graph G is denoted $k(G)$.
    - An undirected graph G is **k-edge-connected** if it remains connected after any k - 1 edges are removed from the graph. The **edge connectivity** of a graph is the largest k such that the graph is k-edge-connected. The edge connectivity of a graph G is denoted $\lambda(G)$.

6. **Euler circuits and trails**

    - An **Euler circuit** in an undirected graph is a circuit that contains every edge and every vertex. Note that a circuit, by definition, has no repeated edges, so an Euler circuit contains each edge exactly once.
    - If an undirected graph G has an Euler circuit, then G is connected and every vertex in G has an even degree.
    - If an undirected graph G is connected and every vertex in G has an even degree, then G has an Euler circuit.
    - An undirected graph G has an Euler circuit if and only if G is connected and every vertex in G has even degree.
    - An **Euler trail** is an open trail that includes each edge. Note that a trail, by definition, has no repeated edges, so an Euler trail contains each edge exactly once. In an open trail, the first and last vertices are not equal. As with Euler circuits, there is a simple set of conditions that characterize when an undirected graph has an Euler trail.
    - An undirected graph G has an Euler trail if and only if G is connected and has exactly two vertices with odd degree.

7. **Hamiltonian cycles and paths**

- A **Hamiltonian cycle** in an undirected graph is a cycle that includes every vertex in the graph. Note that a cycle, by definition, has no repeated vertices or edges, except for the vertex which is at the beginning and end of the cycle. Therefore, every vertex in the graph appears exactly once in a Hamiltonian cycle, except for the vertex which is at the beginning and end of the cycle.
- A **Hamiltonian path** in an undirected graph is a path that includes every vertex in the graph. Note that a path, by definition, has no repeated vertices or edges, so every vertex appears exactly once in a Hamiltonian path.

Closed walk:

⟨ x, v, w, y, z, x ⟩

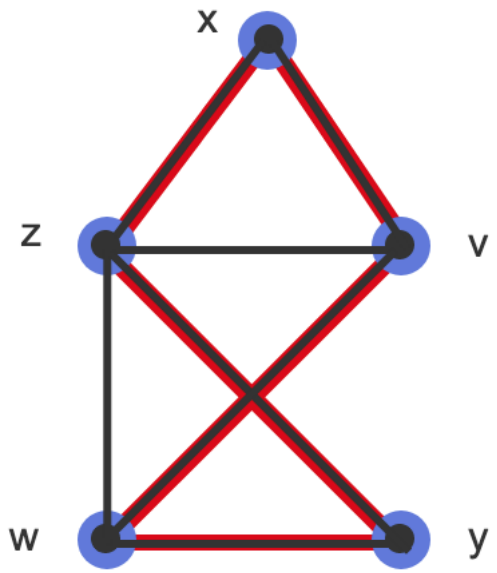
Closed walk is a cycle:

No edge repeated

No vertex repeated, except first and last

Cycle is a Hamiltonian cycle:

Every vertex reached



Open walk:

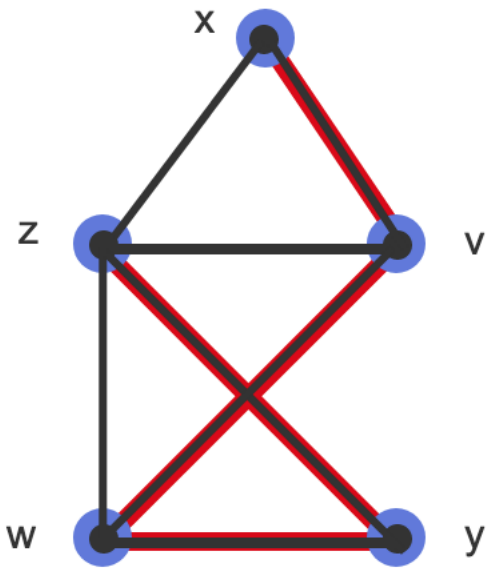
⟨ x, v, w, y, z ⟩

Open walk is a path:

No edge or vertex repeated

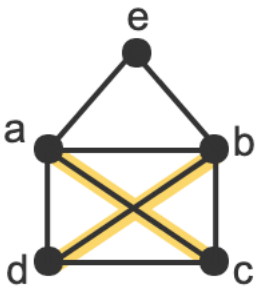
Path is a Hamiltonian path:

Every vertex reached



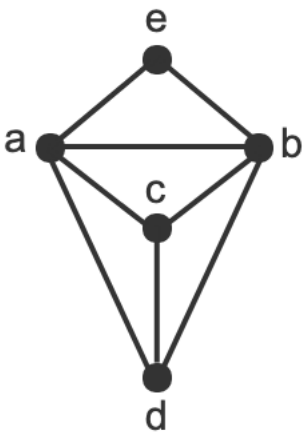
8. **Planar graphs**

- Graph G
  V = { a, b, c, d, e }
  E = { { a, b }, { b, c }, { c, d }, { d, a }, { a, c }, { b, d }, { e, a }, { e, b } }



An embedding of graph G.

The embedding is not planar.



A different embedding of graph G.

The embedding is planar.

(No edged crossing)

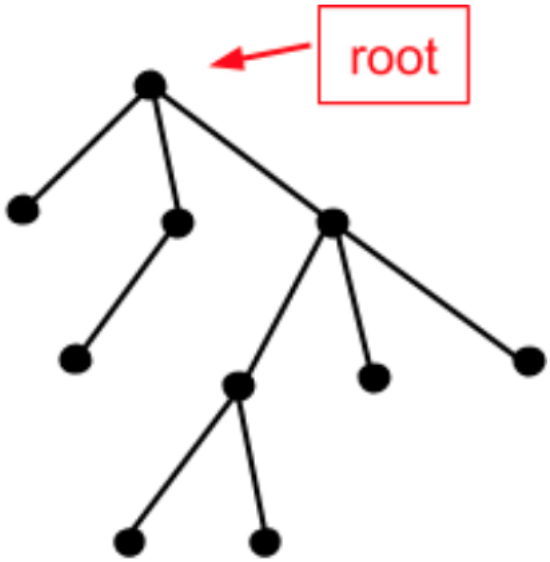
9. **Graph coloring**

- Let G = (V, E) be an undirected graph and C a finite set of colors. A **valid coloring** of G is a function f: V $\rightarrow$ C such that for every edge {x, y} $\in$ E, $f(x) \neq f(y)$. If the size of the range of function f is k, then f is called a **k-coloring** of G.
- The **greedy coloring algorithm**:
- Number the set of possible colors. Assume that there is a very large supply of different colors, even though they might not all be used.
- Order the vertices in any arbitrary order.
- Consider each vertex v in order:
  - Assign v a color that is different from the color of v's neighbors that have already been assigned a color. When selecting a color for v, use the lowest numbered color possible.

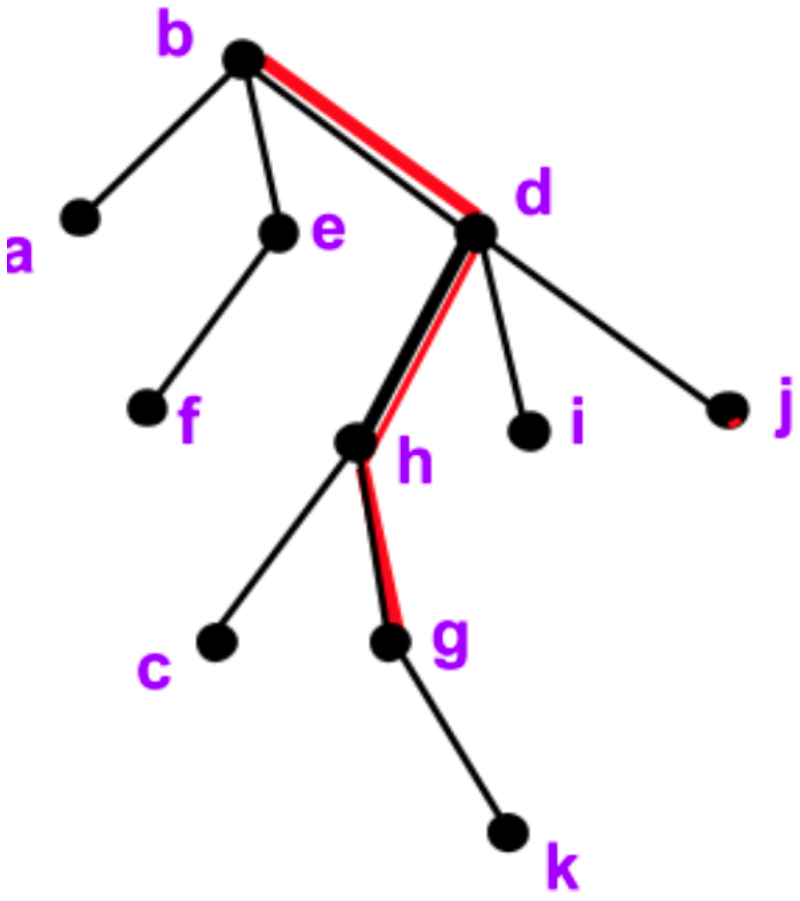
Trees

1. **Introduction to trees** - A **tree** is an undirected graph that is connected and has no cycles. The tree on the left is called a **free tree** because there is no particular organization of the vertices and edges. The tree on the right is called a **rooted tree**. The vertex at the top of the drawing is designated as the **root** of the tree. The remaining vertices are organized according to their distance from the root. The distance between two vertices in an undirected graph is the number of edges in the shortest path between the two vertices. The **level** of a vertex is its distance from the root. The **height** of a tree is the highest level of any vertex. The tree on the right has height 3.

A free tree



A rooted tree

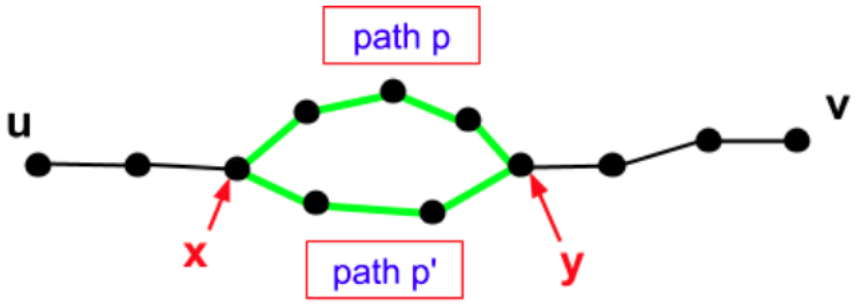


- Every vertex in a rooted tree T has a unique **parent**, except for the root which does not have a parent. The parent of vertex v is the first vertex after v encountered along the path from v to the root. (Ex: The parent of vertex g is h.)
- Every vertex along the path from v to the root (except for the vertex v itself) is an ancestor of vertex v. (Ex: The **ancestors** of vertex g are h, d, and b.)
- If v is the parent of vertex u, then u is a **child** of vertex v. (Ex: Vertices c and g are the children of vertex h.)
- If u is an ancestor of v, then v is a **descendant** of u. (Ex: The descendants of vertex h are c, g, and k.)
- A **leaf** is a vertex which has no children. (Ex: The leaves are a, f, c, k, i, and j.)
- Two vertices are **siblings** if they have the same parent. (Ex: Vertices h, i, and j are siblings because they have the same parent, which is vertex d.)
- A **subtree** rooted at vertex v is the tree consisting of v and all v's descendants. (Ex: The subtree rooted at h includes h, c, g, and k and the edges between them.)

2. **Tree application examples**

3. **Properties of trees** - The definitions for parent and child vertices in rooted trees make use of the fact that every vertex has a unique path to the root. There is, in fact, a unique path between any pair of vertices in a tree. The theorem below applies to free trees as well as rooted trees. **Theorem**: There is a unique path between every pair of vertices in a tree.

    - **Proof**. A tree is defined to be a connected graph with no cycles. Because every tree is connected, there is at least one path between every pair of vertices. It remains to establish that there is at most one path between every pair of vertices. The proof is by contrapositive. We assume that there is a pair of vertices u and v such that there are two distinct paths between u and v and then show that the graph must have a cycle (and therefore can not be a tree).

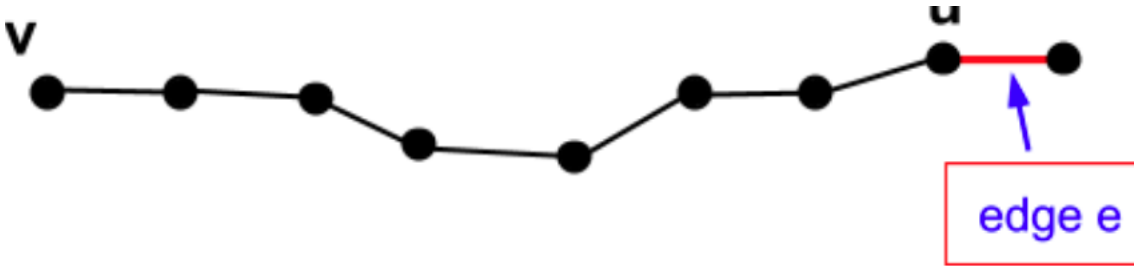


    - Let p and p' be the two distinct paths between u and v. Find the first place where the two paths differ. Let x be the vertex just before the point where the two paths diverge. Follow path p until it hits a vertex y that is also contained in p'. (Both paths end up at v, so there has to be a point where the two paths come together). The portion of the path p between x and y and the portion of the path p' between x and y form a cycle.

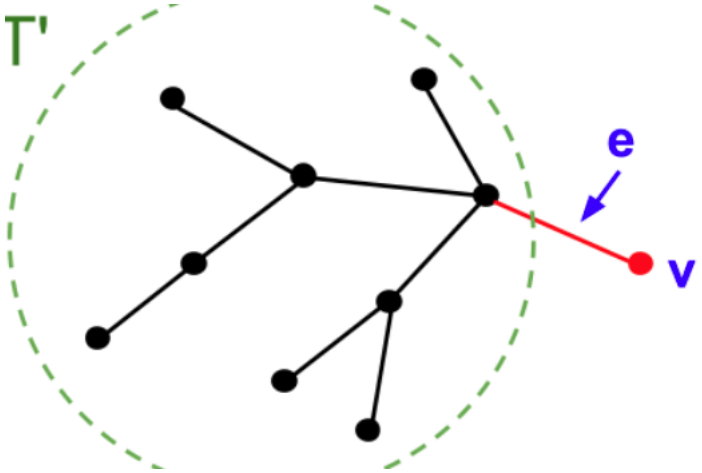
- The term leaf is defined in rooted trees as a vertex with no children. A leaf has a natural definition in free trees as well. A leaf of a free tree is a vertex of degree 1. There is one technicality: if a free tree has only one vertex, then that vertex is a leaf. A vertex is an internal vertex if the vertex has degree at least two. The following theorem gives a lower bound on the number of leaves in a free tree:

  Theorem: Any free tree with at least two vertices has at least two leaves.

- **Proof**. Find the longest path p in the tree. Let u and v be the first and last vertices in the path. Suppose u is not a degree 1 vertex. (The same argument will hold for v). Then there is an edge e that is incident to u and not part of the path p.



- If the other endpoint of e, that is not vertex u, is part of the path p, then the graph has a cycle and is not a tree. The path p can be extended by adding e along with the other endpoint of e. The fact that the path p can be extended contradicts the assumption that p was the longest path in the tree.
- **Theorem**: Let T be a tree with n vertices and m edges, then m = n - 1
- **Proof**. The proof is by induction on the number of vertices. The base case is where n = 1. If T has one vertex, then it is has no edges. Then m = 0 = n - 1.
- For the inductive step, assume the theorem holds for trees with n - 1 vertices and prove that it holds for trees with n vertices. Consider an arbitrary tree T with n vertices. Since n $\leq$ 2, by the previous theorem, the tree has at least two leaves. Let v be one of the leaves. Remove v from T along with the edge e incident to v. The resulting graph (call it T') is also a tree and has n-1 vertices.



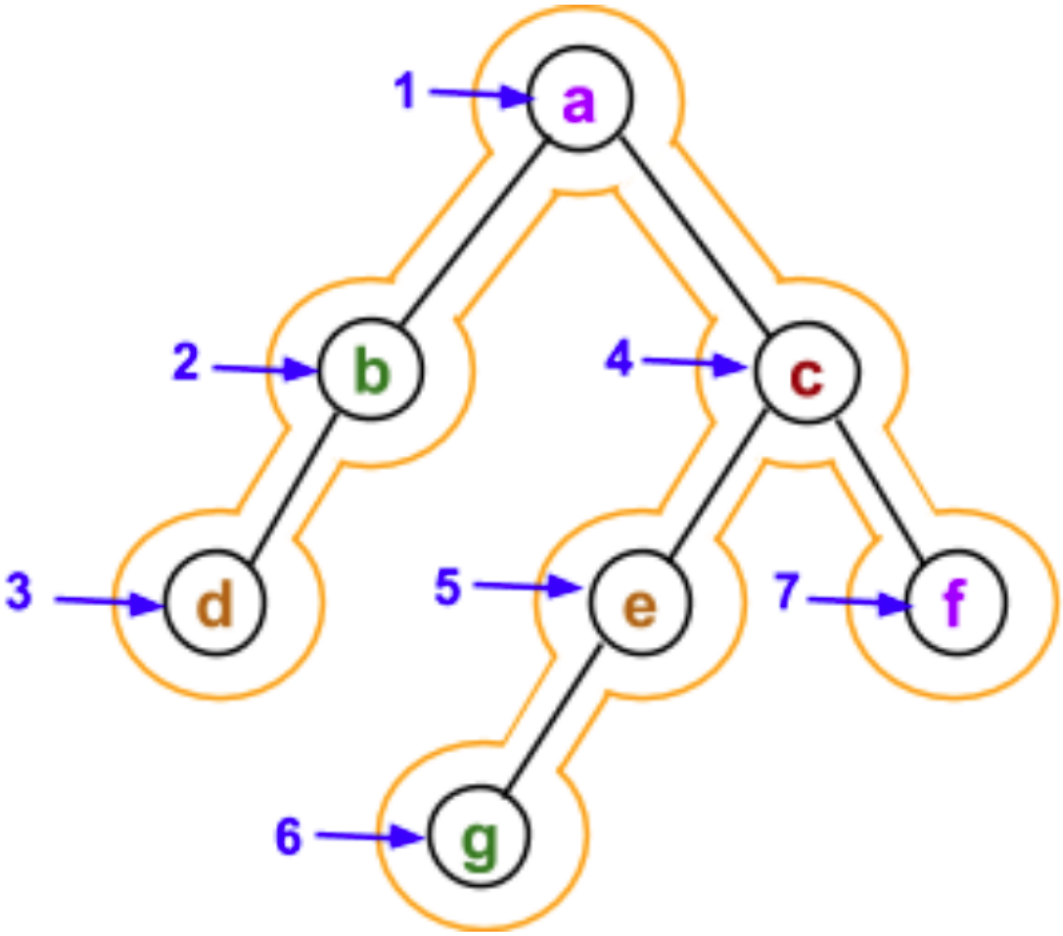
- By the induction hypothesis, The number of edges in T' is (n - 1) - 1 = n - 2. T has exactly one more edge than T', because only edge e was removed from T to get T'. Therefore the number of edges in T is n - 2 + 1 = n - 1

4. **Tree traversals**

- Pseudocode for pre-order traversal.

```
Pre-order(v)

process(v)
For every child w of v:
    Pre-order(w)
End-for
```



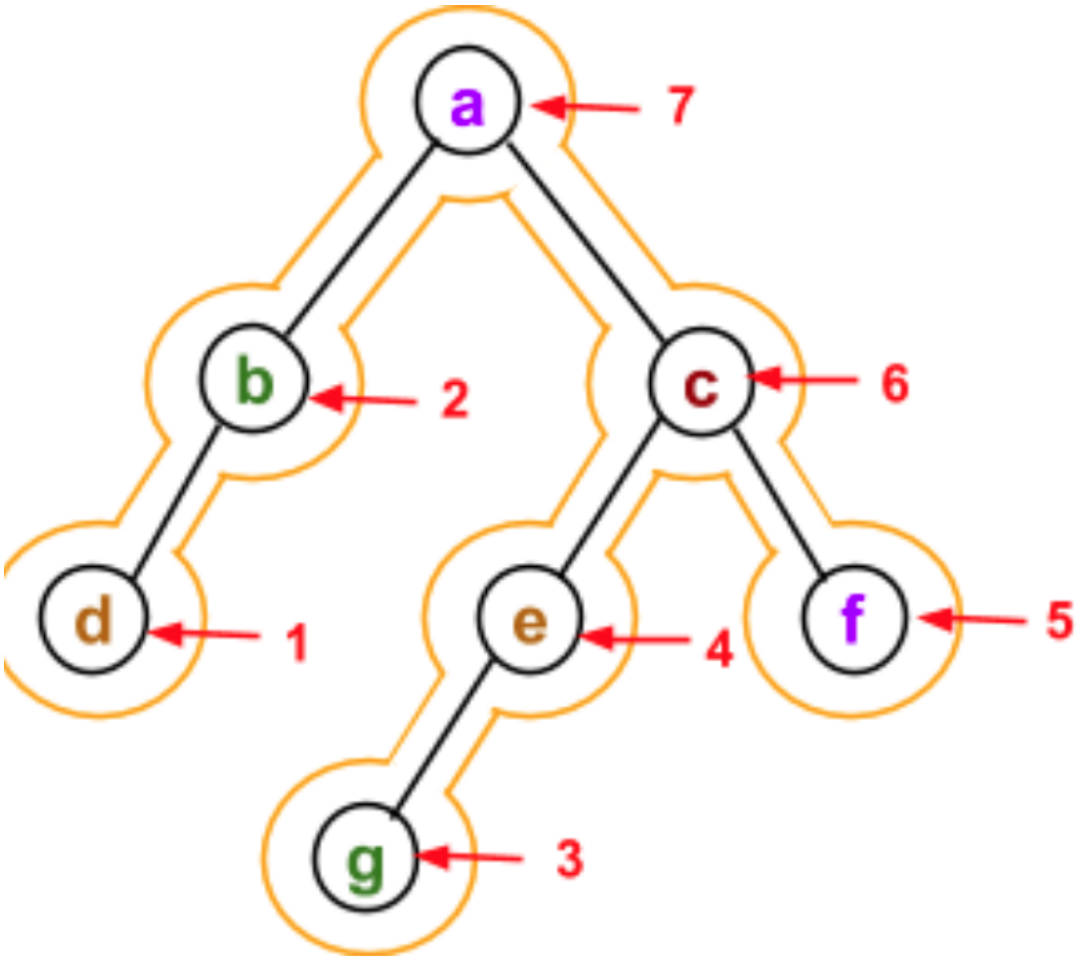
Pre-order traversal:

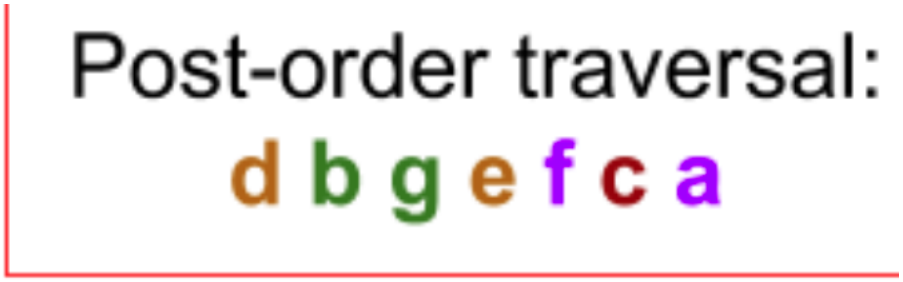
a b d c e g f

- Pseudocode for post-order traversal.

```
Post-order(v)

For every child w of v:
    Post-order(w)
End-for
process(v)
```





5. **Spanning trees and graph traversals**

   - A spanning tree of a connected graph G is a subgraph of G which contains all the vertices in G and is a tree.
   - **Depth-First Search**
   - **Breadth-first-search**

6. **Minimum spanning trees**: A spanning tree of a graph specifies a subset of the edges that provides connectivity between every pair of vertices while minimizing the number of edges included. Every spanning tree of a graph with n nodes has n-1 edges, so there is no reason to prefer one spanning tree over another if the only goal is to include as few edges as possible.

   - A **weighted graph** is a graph G = (V, E), along with a function w: E $\rightarrow$ R. The function w assigns a real number to every edge.
   - A **minimum spanning tree (MST)** of a weighted graph, is a spanning tree T of G whose weight is no larger than any other spanning tree of G.
   - A graph can have more than one minimum spanning tree because there can be more than one spanning tree with the same weight. In the extreme case, if the weight of every edge is 1, then every spanning tree has weight n - 1 and every spanning tree is also a minimum spanning tree.
   - **Prim's algorithm** finds a minimum spanning tree