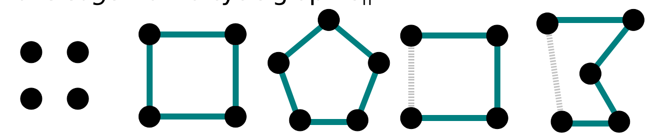
Graph Theory

- **Outline**:
  - Introduction
    - Basic terminology and concepts
    - Representations of graphs
    - Operations on Graphs
  - Walks, trails, paths, circuits and cycles
    - Eulerian circuits
    - Hamiltonian cycles
  - **Special types of graphs**
  - Graph isomorphism and homeomorphism
  - Trees
  - Graph problems and their applications

Special Types of Graphs

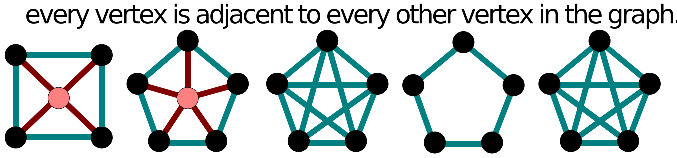
- **trivial graph** ≡ one vertex + no edges.
- **null graph**, N_n, ≡ n vertices + no edges.
- **empty graph** ≡ no vertices + no edges.
- **cycle graph**, C_n, ≡ n vertices + edges form a cycle of length n.
- **path graph**, P_n, ≡ n vertices + obtained by *removing one edge* from a cycle graph C_n.



N_4 **C_4** **C_5** **P_4** **P_5**

Special Types of Graphs

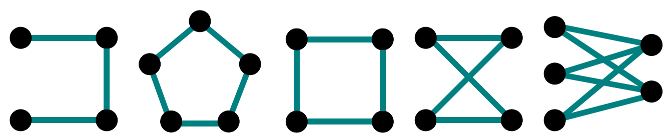
- **wheel graph**, W_n, ≡ n vertices + obtained by *adding a new vertex*, called the **hub**, to a C_{n-1} cycle graph and joining that new vertex to all n-1 vertices in the C_{n-1} graph.
- **k-regular graph** ≡ *simple* graph + every vertex has a degree of k.
- **complete graph** K_n, ≡ *simple* graph + n vertices + every vertex is adjacent to every other vertex in the graph.



W_5 **W_6** **4-regular** **2-regular** **K_5**

Special Types of Graphs

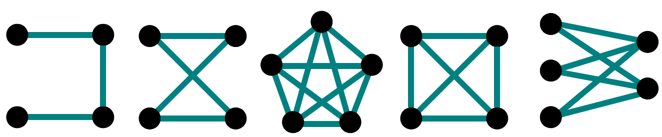
- **bipartite graph** ≡ vertices can be *partitioned into two sets* so that every edge in the graph only joins one vertex in one set to a vertex in the other set.
  - *A graph is bipartite if all the cycles it contains are of even length.*
- **complete bipartite graph**, $K_{m,n}$ ≡ simple bipartite graph where every vertex in one set is adjacent to every vertex in the other set.



P_4 **2-regular** **W_5** **$K_{2,2}$** **$K_{3,2}$**

Special Types of Graphs

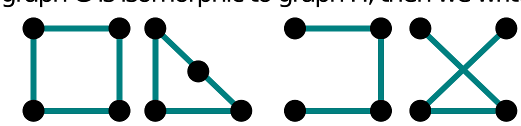
- **planar graph** ≡ graph which can be drawn on a plane such that its edges *do not cross* each other.



P_4 **$K_{2,2}$** **K_5** **3-regular** **$K_{3,2}$**

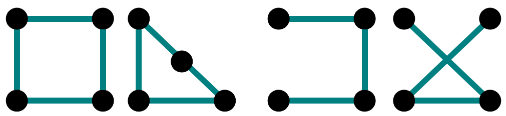
Isomorphism

- **Graph G is isomorphic to a graph H** if there exists a one-to-one correspondence ϕ between from V(G) to V(H) such that ϕ preserves adjacency, that is, if $(u,v) \in E(G)$ iff $(\phi(u), \phi(v)) \in E(H)$.
  - one-to-one and onto correspondence between V(G) and V(H) is just a re-labeling of the vertices.
  - if graph G is isomorphic to graph H, then we write $G \cong H$.

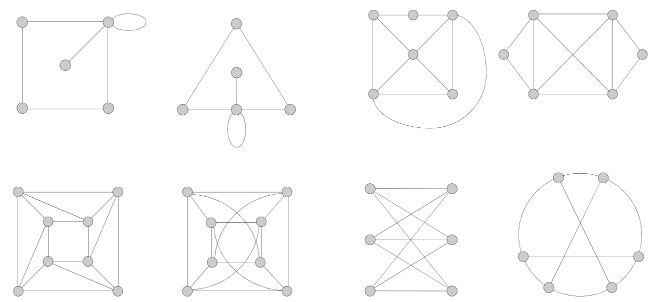


Isomorphism

- THEOREM. Two simple graphs G and H are isomorphic if and only if for some ordering of their respective vertices, their *adjacency matrices are equal*.
- If graphs G and H are not isomorphic, an **invariant** is a property of G that H does not have (or *vice versa*).
- graph G is **self-complementary** if it is isomorphic to its complement G^c.

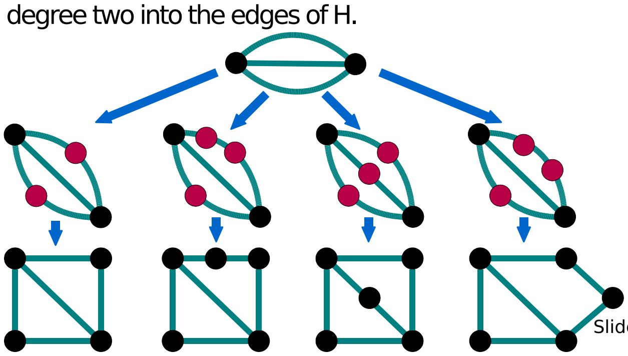


Isomorphism



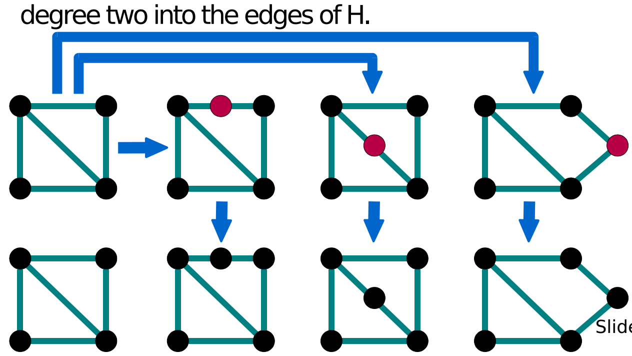
Homeomorphism

- **Graphs G1 and G2 are homeomorphic** if they can be obtained from a graph H by inserting new vertices of degree two into the edges of H.



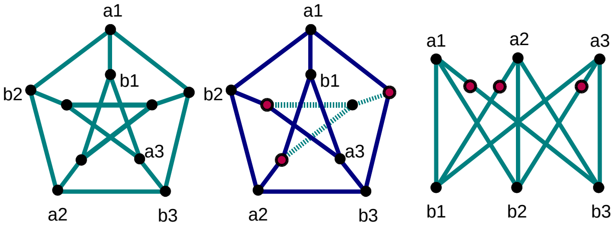
Homeomorphism

- **Graphs G1 and G2 are homeomorphic** if they can be obtained from a graph H by inserting new vertices of degree two into the edges of H.



Homeomorphism

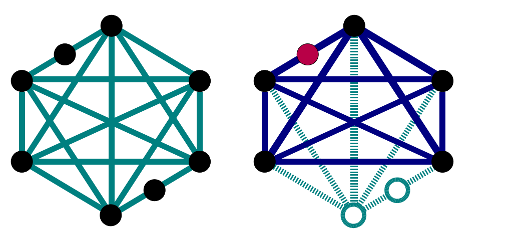
- **KURATOWSKI's THEOREM**.
  A graph is planar if and only if it does not have a subgraph that is homeomorphic to the complete graph K_5 or to the complete bipartite graph $K_{3,3}$.



Petersen's Graph

Homeomorphism

- **KURATOWSKI's THEOREM**.
  A graph is planar if and only if it does not have a subgraph that is homeomorphic to the complete graph K_5 or to the complete bipartite graph $K_{3,3}$.

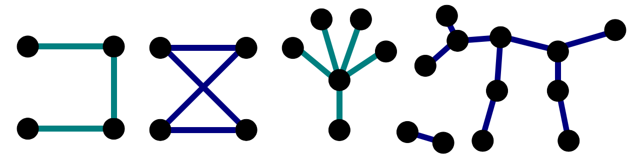


Trees

- **TREES** are *special graphs* make up a subclass of graphs that are widely used.
- Some applications:
  - **Linguistics**:
    - used to analyze sentence structure
  - **Medicine**:
    - decision trees are used to form a diagnosis based on a whole array of symptoms and test results.
  - **Computer Science**:
    - data can be stored in tree data structures for easy access.
    - used to organize and relate data in a database.

Trees

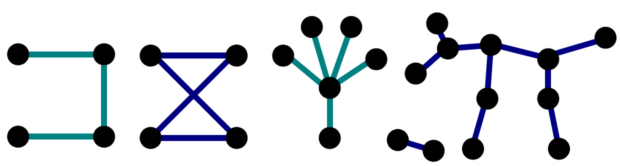
- A **tree** ≡ acyclic connected graph
- A **forest** ≡ set of trees OR a graph whose components are trees
- THEOREM. In a tree, any two vertices are connected by a unique path.
- THEOREM. If graph G is a tree, then |E(G)| = |V(G)| - 1.



P_4 $K_{2,2}$

Trees

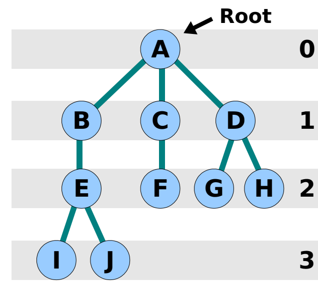
- COROLLARY. Every non-trivial tree has at least two vertices of degree 1.
- THEOREM. A connected graph is a tree iff *every* edge is a cut edge.
- THEOREM. In a tree, a vertex v is a cut-vertex iff $\rho(v)>1$.



P_4 $K_{2,2}$

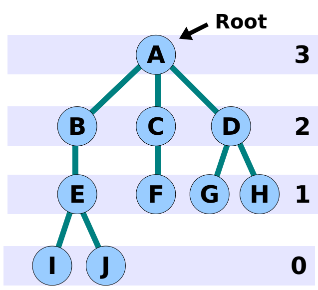
Trees

- **nodes** ≡ vertices in a tree
- **leaf** ≡ end vertex in a tree.
- **interior node** ≡ vertex in a tree that is not a leaf.
- **rooted tree** ≡ tree + vertex r designated as root of tree
- **level** of a node v ≡ length of the path from root to node v. [The level of the root is zero(0).]



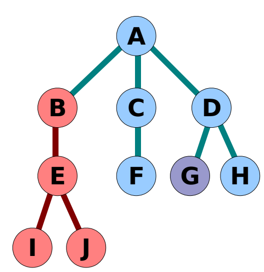
Trees

- **height** of the tree ≡ *greatest level assignment* given to a node in the tree
  - **height of the root** ≡ height of the tree
  - **height of a node v** ≡ one less than its parent
- Can also define
  - siblings
  - parent
  - child
  - ancestors
  - descendants



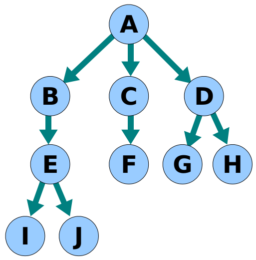
Trees

- **subtree rooted at node v** of a tree ≡ *subgraph induced* by the node v and all of its descendants (if any).



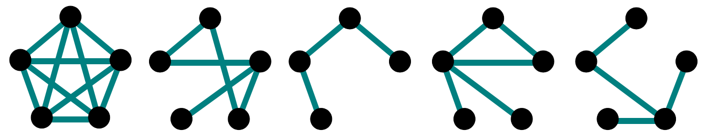
Trees

- **subtree rooted at node v** of a tree ≡ *subgraph induced* by the node v and all of its descendants (if any).
- **directed tree** ≡ tree which contains a directed path from the root to each vertex.



Trees

- A **spanning tree** of a graph G is a *spanning subgraph* T of G such that T *is a tree*.
- COROLLARY. Every connected graph contains a spanning tree.

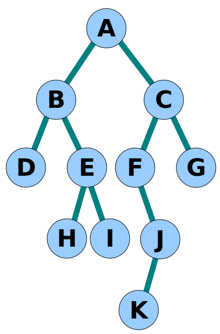


K_5 P_4

Binary Trees

- **binary tree** ≡ a tree where each node has either
  - no children
  - a left child
  - a right child
  - both a left and a right child

NOTE: Distinction is always made between a left child and a right child.

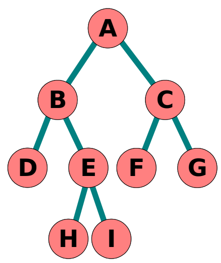


Binary Trees

- **Binary Tree Traversal**
  A systematic way of listing down the nodes of a binary tree.

Suppose T is a binary tree with root r whose left and right subtrees are T_L and T_R, respectively.

- **Preorder Listing/Traversal**
  root r
  + nodes in T_L *in preorder*
  + nodes of T_R *in preorder.*

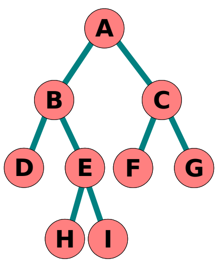


Binary Trees

- **Binary Tree Traversal**
  A systematic way of listing down the nodes of a binary tree.

Suppose T is a binary tree with root r whose left and right subtrees are T_L and T_R, respectively.

- **Postorder Listing/Traversal**
  nodes in T_L *in postorder*
  + nodes of T_R *in preorder*
  + root r

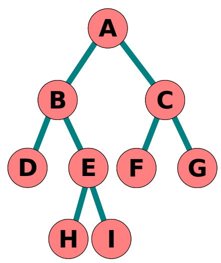


Binary Trees

- **Binary Tree Traversal**
  A systematic way of listing down the nodes of a binary tree.

Suppose T is a binary tree with root r whose left and right subtrees are T_L and T_R, respectively.

- **Inorder Listing/Traversal**
  nodes in T_L *in inorder*
  + root r
  + nodes of T_R *in inorder.*



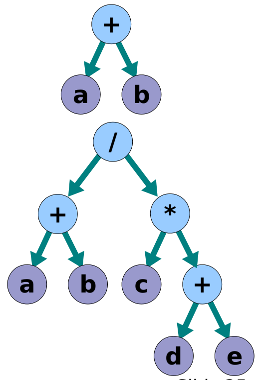
Binary Trees

- **Expression Trees**
  - ≡ *directed* binary trees where
    - each interior node contains an operator
    - each leaf contains an operand.

***Examples*:**

Draw the expression tree representing each of the following:

- a+b
- (a+b)/(c*(d+e))



Binary Trees

- **Different forms of the expression**
  - **Infix form**
    - results from an inorder traversal
    - operator is written between its operands.
  - **Prefix form**
    - results from a preorder traversal
    - operator is written before its operands.
  - **Postfix form**
    - results from a postorder traversal
    - operator is written after its operands.