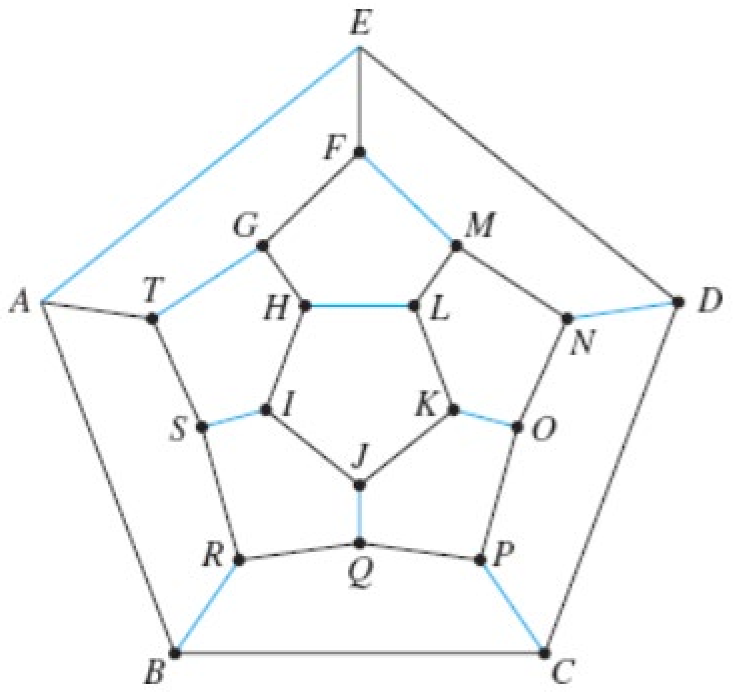
CS 315 - Day 20, Graphs and Trees

In-class w/***assigned groups of four (4)***, write 0. and 1. on board as ground rule(s):

0. No devices.
1. Intro: Name, POE, Icebreaker.
2. Pick someone on the group who will report out.
3. Can you find a circuit in the graph below where all the ***vertices*** (except the first and last) appear exactly once?

Note that you are not required to use all the edges.



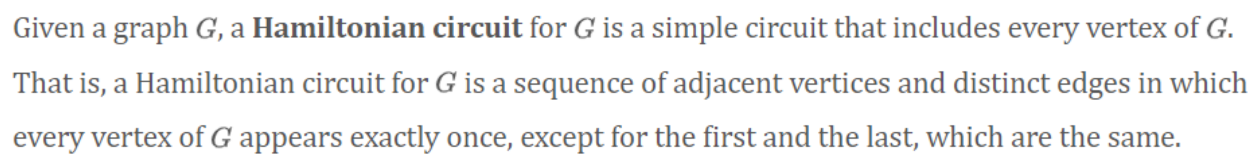
4. Does this graph have an Euler Circuit? If so, give the Euler Circuit, if not, give a reason why not.

3. ***YES***, this does have a circuit satisfying the above conditions:

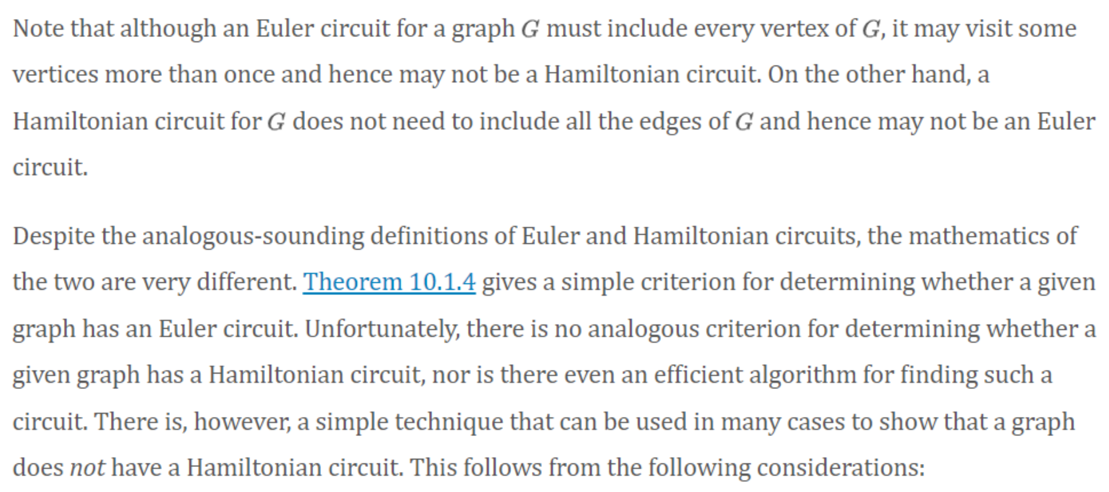
$$A\ B\ C\ D\ E\ F\ G\ H\ I\ J\ K\ L\ M\ N\ O\ P\ Q\ R\ S\ T\ A$$

We call this a ***Hamiltonian Circuit*** (HC).

4. ***NO***, this graph does not have an Euler Circuit, several vertices have odd degree, including A, B, and C.

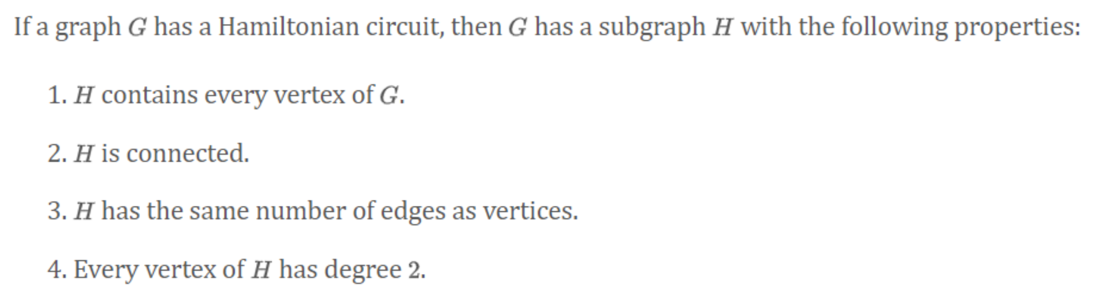


Given a graph G, a **Hamiltonian circuit** for G is a simple circuit that includes every vertex of G. That is, a Hamiltonian circuit for G is a sequence of adjacent vertices and distinct edges in which every vertex of G appears exactly once, except for the first and the last, which are the same.



Note that although an Euler circuit for a graph G must include every vertex of G, it may visit some vertices more than once and hence may not be a Hamiltonian circuit. On the other hand, a Hamiltonian circuit for G does not need to include all the edges of G and hence may not be an Euler circuit.

Despite the analogous-sounding definitions of Euler and Hamiltonian circuits, the mathematics of the two are very different. Theorem 10.1.4 gives a simple criterion for determining whether a given graph has an Euler circuit. Unfortunately, there is no analogous criterion for determining whether a given graph has a Hamiltonian circuit, nor is there even an efficient algorithm for finding such a circuit. There is, however, a simple technique that can be used in many cases to show that a graph does *not* have a Hamiltonian circuit. This follows from the following considerations:

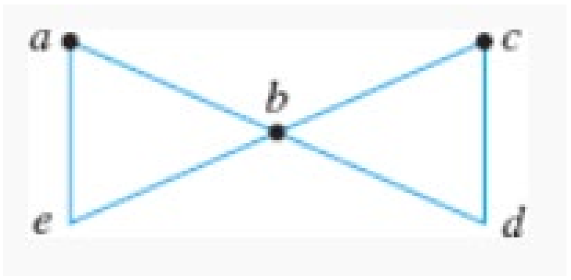


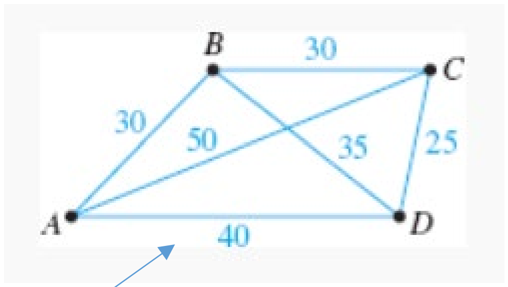
If a graph G has a Hamiltonian circuit, then G has a subgraph H with the following properties:

1. H contains every vertex of G.
2. H is connected.
3. H has the same number of edges as vertices.
4. Every vertex of H has degree 2.

Finding a HC is constructive. Start with just the vertices of the graph. In the HC, each of these vertices will have
$deg = 2$, so include the edges that are required, those that are incident on a vertex of $deg = 2$.

5. So, do the following graphs have HC's? If so, give the HC, and if not, give a reason why:





No, it does not. The edges connecting a and b, a and e, e and b are required, as are the edges connecting c and b, c and d, d and b are also required. So in the HC, vertex b would have deg 4, which isn't allowed.

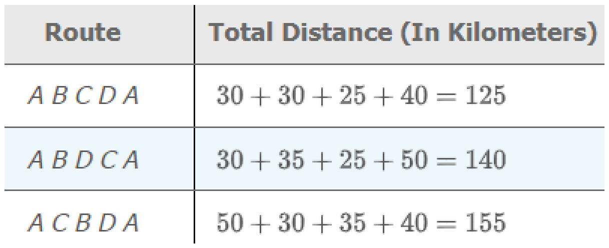
Yes, A, B, C, D, A is one.

For practical applications of this graph theory, let's consider how HC's help us with the Travelling Salesperson Problem.

In the above graph, the edges are weighted. Suppose a salesperson is to visit each city exactly once, starting and ending at A. Which route will minimize the total distance travelled?

This question is equivalent to asking what is the HC with minimum weight?

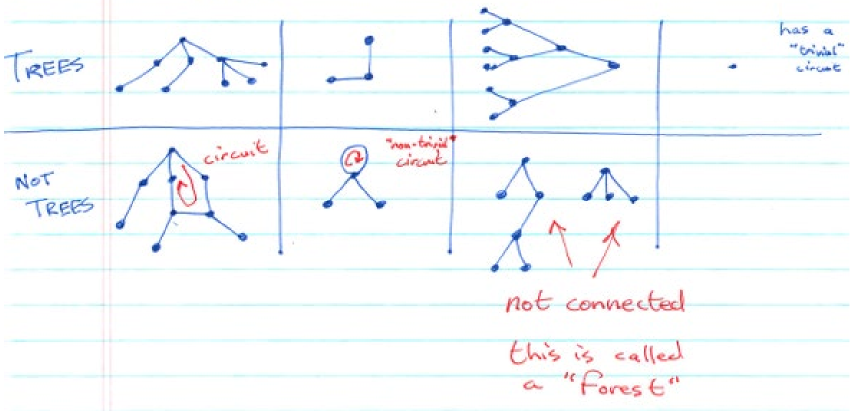
Since we are starting and ending at A, and the graph is complete (all vertices have edges to each other vertex), we have 3 choices for the next node (B, C, or D) , then two choices, then one, or $3! = 6$ choices. But we can disregard half of these, they would have the same weight but just travelled in reverse order, like A, B, C, D, A vs. A, D, C, B, A



Route	Total Distance (In Kilometers)
A B C D A	$30 + 30 + 25 + 40 = 125$
A B D C A	$30 + 35 + 25 + 50 = 140$
A C B D A	$50 + 30 + 35 + 40 = 155$

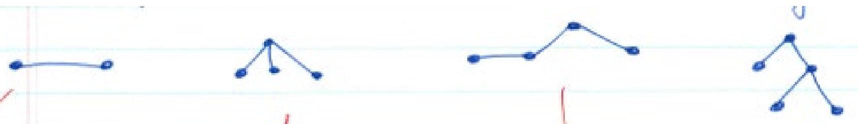
Trees are special graphs.

A ~~tree~~ graph is a tree if and only-if it is connected + has no non-trivial circuits



A vertex of degree = 1 is called a terminal vertex or leaf

A vertex w/ deg > 1 is called an internal vertex or branch vertex



Important Tree Thm ① For any positive integer n, any tree with n vertices has n-1 edges

n=2 vertices
1 edge

4 vertices
3 edges

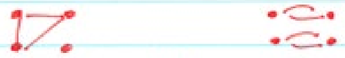
4 vertices
3 edges

5 vertices
4 edges

③

Q: So, what would a graph with n vertices + n-1 edges, that is NOT a tree, look like?

n = 4 vertices
3 edges



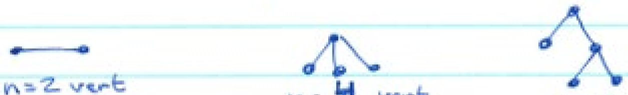
① not connected
② has a circuit

① not connected
② has a circuit

Important Tree Thm ② For any positive integer n, if G is a connected graph w/ n vertices and n-1 edges, then G is a tree.

$\overset{①}{\text{connected}}$ & $\overset{②}{\text{n vert}}$ & $\overset{③}{\text{(n-1) edges}}$ $\Longleftrightarrow$ tree

TD = total degree, the sum of the degrees of all the vertices



n=2 vert
1 edge
TD = 2

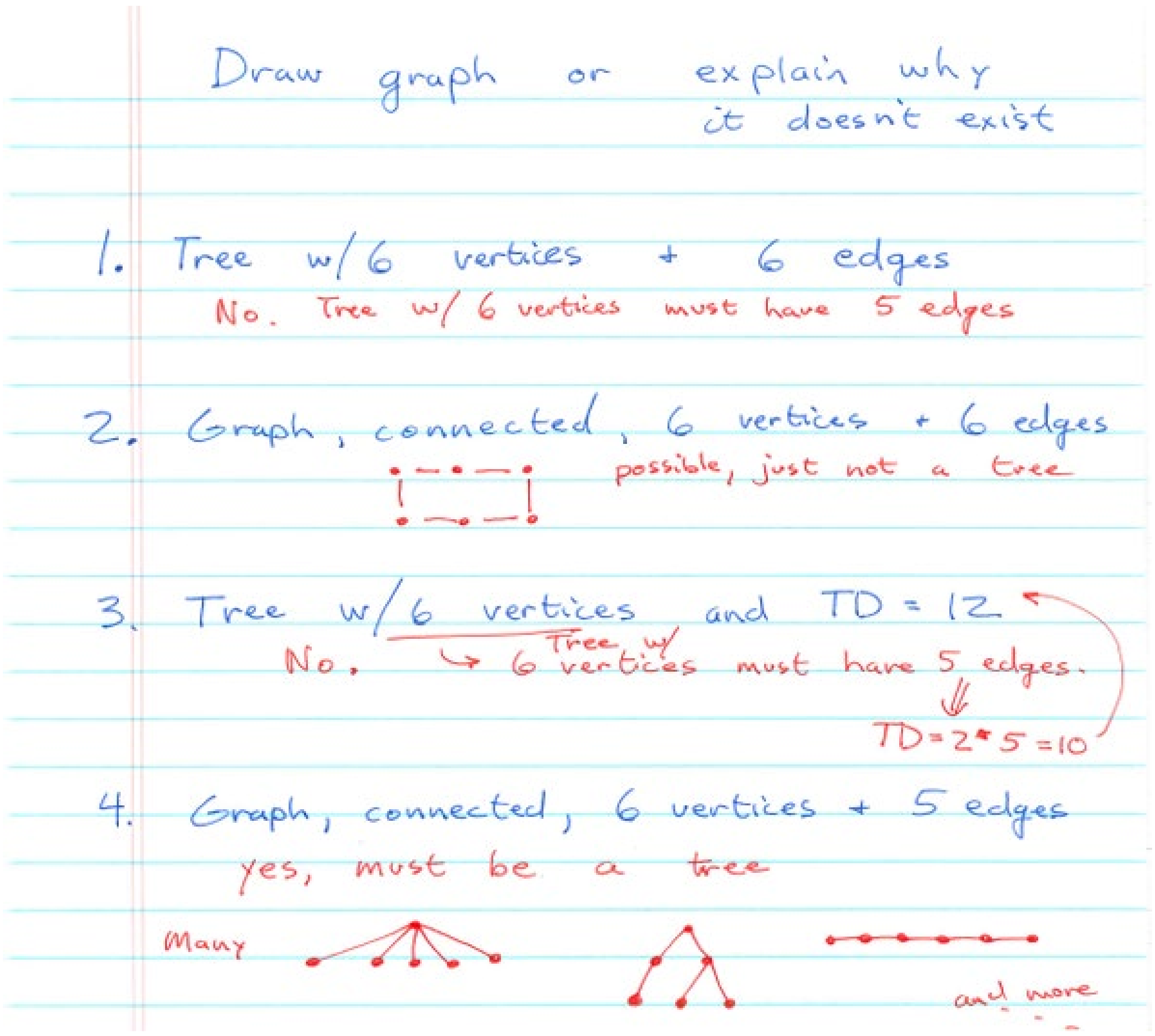
n= 4 vert
3 edges
TD = 6

n = 5 vert
4 edges
TD = 8

- - - -

pattern?
how are # edges + TD related?

in HW



Draw graph or explain why it doesn't exist

1. Tree w/ 6 vertices + 6 edges

No. Tree w/ 6 vertices must have 5 edges

2. Graph, connected, 6 vertices + 6 edges

possible, just not a tree

3. Tree w/ 6 vertices and TD = 12

No. → Tree w/ 6 vertices must have 5 edges.

⇓

TD = 2 * 5 = 10

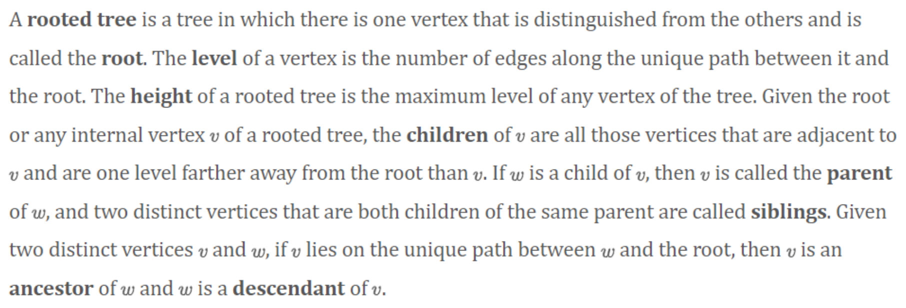
4. Graph, connected, 6 vertices + 5 edges

yes, must be a tree

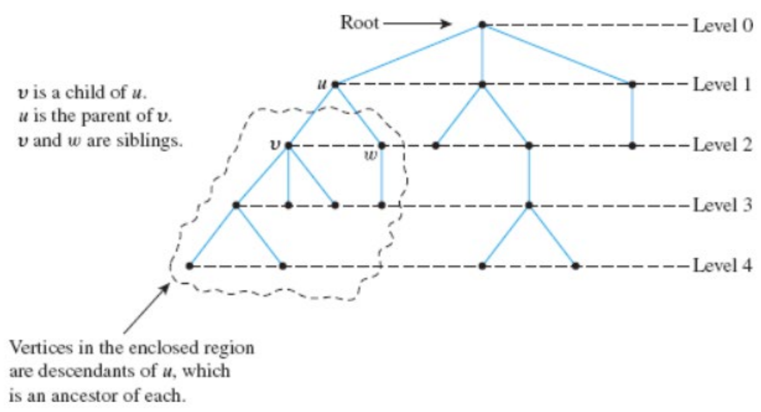
Many

and more

Next, we define a ***rooted tree*** as one where one vertex has been distinguished from the others and is called the ***root***.



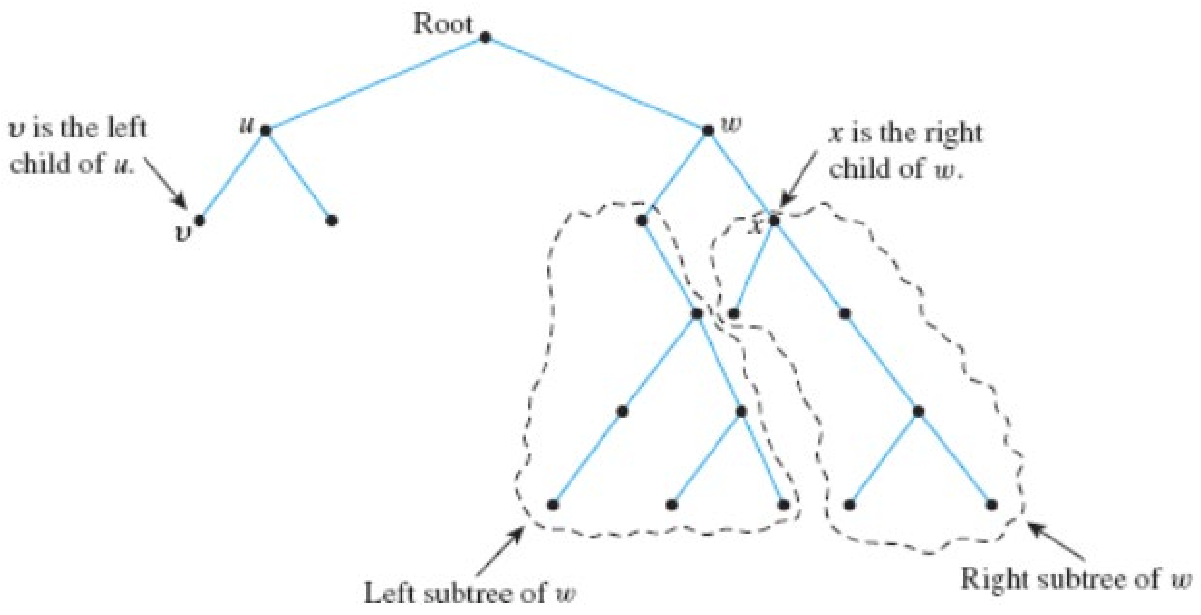
A **rooted tree** is a tree in which there is one vertex that is distinguished from the others and is called the **root**. The **level** of a vertex is the number of edges along the unique path between it and the root. The **height** of a rooted tree is the maximum level of any vertex of the tree. Given the root or any internal vertex v of a rooted tree, the **children** of v are all those vertices that are adjacent to v and are one level farther away from the root than v. If w is a child of v, then v is called the **parent** of w, and two distinct vertices that are both children of the same parent are called **siblings**. Given two distinct vertices v and w, if v lies on the unique path between w and the root, then v is an **ancestor** of w and w is a **descendant** of v.



A **binary tree** is a rooted tree in which every parent has at most two children. Each child in a binary tree is designated either a **left child** or a **right child** (but not both), and every parent has at most one left child and one right child. A **full binary tree** is a binary tree in which each parent has exactly two children.

Given any parent v in a binary tree T, if v has a left child, then the **left subtree** of v is the binary tree whose root is the left child of v, whose vertices consist of the left child of v and all its descendants, and whose edges consist of all those edges of T that connect the vertices of the left subtree. The **right subtree** of v is defined analogously.

Here's an example Binary Tree with those terms:



We use binary trees to represent expressions, they are a key component in compilers and interpreters.

The internal vertices are the operators, and the terminal vertices/leaves are the operands.

The following tree represents the expression $\frac{a}{c+d}$