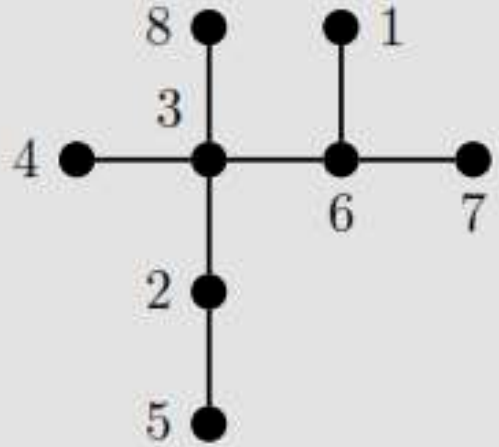
Lec # 16, 17 & 18

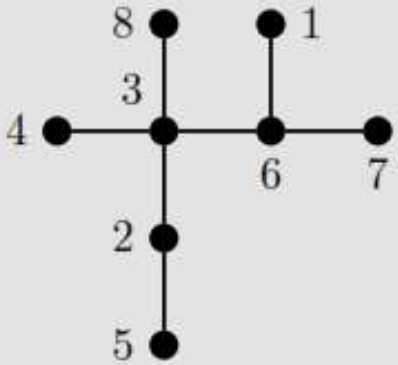
Definition 3.13 Given a tree T on $n > 2$ vertices (labeled $1, 2, \ldots, n$), the ***Prüfer sequence*** of T is a sequence $(s_1, s_2, \ldots, s_{n-2})$ of length $n-2$ defined as follows:

- Let l_1 be the leaf of T with the smallest label.
- Define T_1 to be $T - l_1$.
- For each $i \geq 1$, define $T_{i+1} = T_i - l_{i+1}$, where l_{i+1} is the leaf with the smallest label of T_i.
- Define s_i to be the neighbor of l_i.

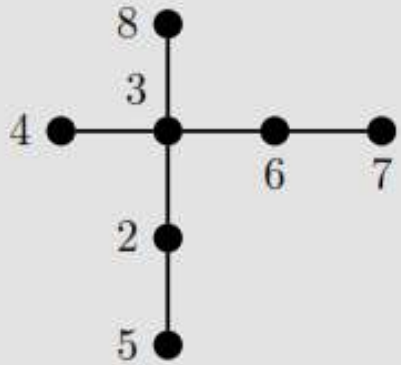
Example 3.4 Find the Prüfer sequence for the tree below.



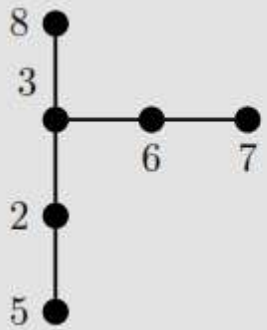
Solution: The pruning of leaves is shown below, with l_i and s_i listed for each step.



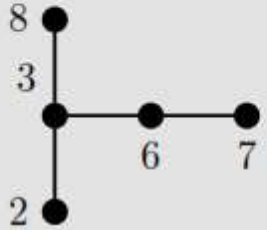
$T:\ l_1 = 1 \quad s_1 = 6$



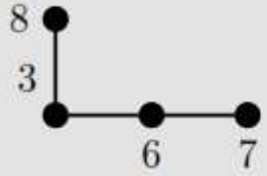
$T_1:\ l_2 = 4 \quad s_2 = 3$



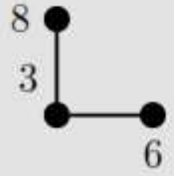
$T_2: \ l_3 = 5 \quad s_3 = 2$



$T_3: \ l_4 = 2 \quad s_4 = 3$



$T_4: \ l_5 = 7 \quad s_5 = 6$



$T_5: \ l_6 = 6 \quad s_6 = 3$

The Prüfer sequence for the tree T is $(6, 3, 2, 3, 6, 3)$.

Example 3.5 Find the tree associated to the Prüfer sequence $(1, 5, 5, 3, 2)$.

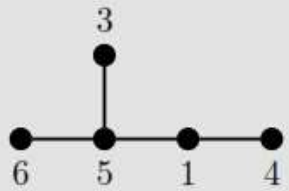
Example 3.5 Find the tree associated to the Prüfer sequence $(1, 5, 5, 3, 2)$.

Solution: First note that the sequence is of length 5, so the tree must have 7 vertices. At every stage we will consider the possible leaves of a subtree created by the earlier pruning. Initially our set of leaves is $L = \{4, 5, 7\}$, since these do not appear in the sequence and so must be the leaves of the full tree. To begin building the tree, we look for the smallest value not appearing in the sequence, namely 4. This must be adjacent to 1 by entry s_1, as shown on the next page.

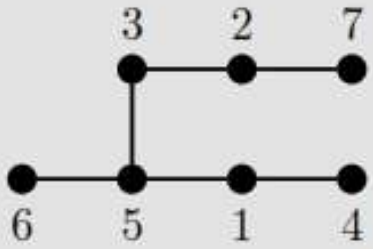
Next, we remove s_1 from the Prüfer sequence and consider the subsequence $(5, 5, 3, 2)$. Now our set of leaves is $L = \{1, 6, 7\}$ since 4 has already been placed as a leaf and 1 is no longer appearing in the sequence. Since 1 is the smallest value in our set L, we know it must be adjacent to $s_2 = 5$, as shown below.

Repeating this process, we see that our subsequence is $(5, 3, 2)$ and $L = \{6, 7\}$. Thus 6 is the leaf adjacent to 5. See the graph below.

Our next subsequence is $(3, 2)$ and $L = \{5, 7\}$, meaning 5 is the smallest value not appearing in the sequence that hasn't already been placed as a "leaf," so there must be and edge from 5 to 3.



Finally we are left with the single entry (2). We know that $L = \{3, 7\}$ and both of these must be adjacent to 2.



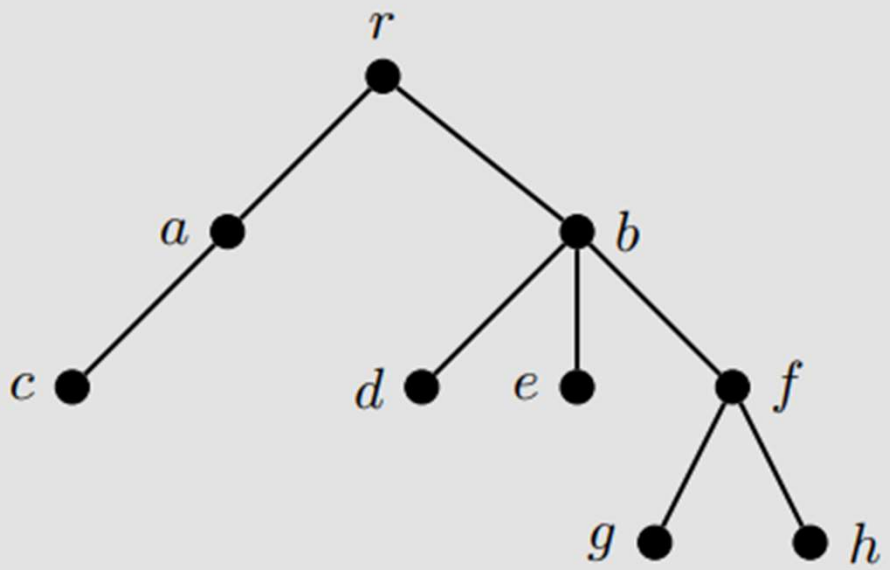
To verify we have the correct tree, we can use the process shown in Example 3.4 above to find the Prüfer sequence of our final tree and verify it matches the one given.

Theorem 3.14 (Cayley's Theorem) There are n^{n-2} different labeled trees on n vertices.

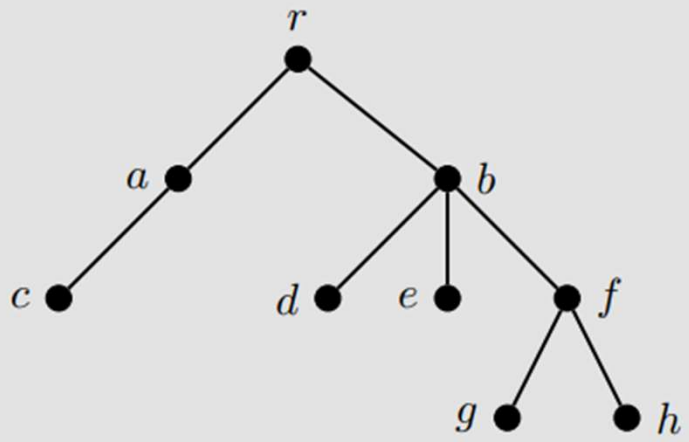
Proof: First, note that there are n^{n-2} possible sequences of length $n-2$ since each spot on the sequence has n options. Each of these sequences can be viewed as a Prüfer sequence and will uniquely determine a tree as shown above. Thus there are n^{n-2} different labeled trees on n vertices.

Definition 3.15 A ***rooted tree*** is a tree T with a special designated vertex r, called the ***root***. The ***level*** of any vertex in T is defined as the length of its shortest path to r. The ***height*** of a rooted tree is the largest level for any vertex in T.

Example 3.6 Find the level of each vertex and the height of the rooted tree shown below.



Example 3.6 Find the level of each vertex and the height of the rooted tree shown below.

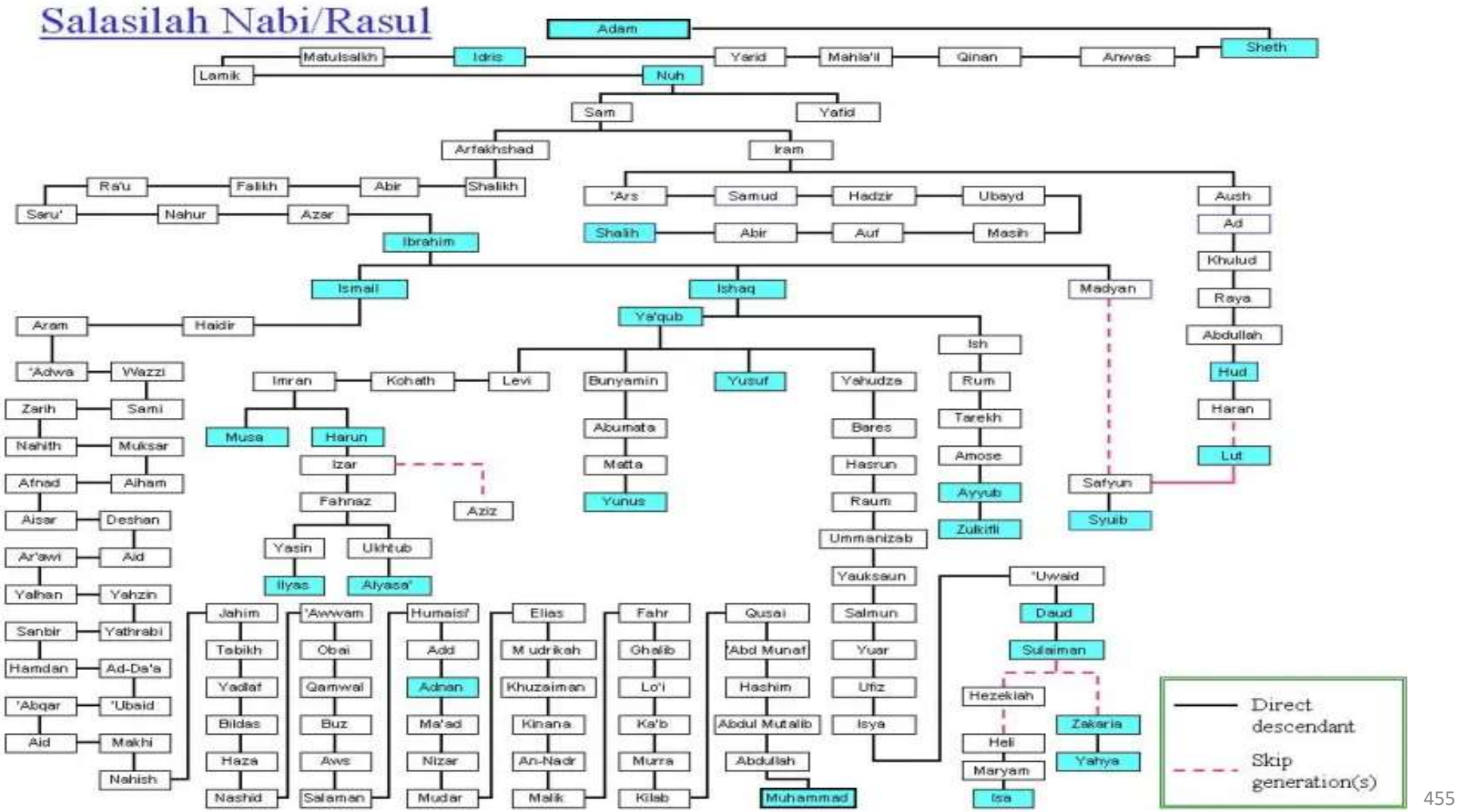


Solution: Vertices a and b are of level 1, c, d, e, and f of level 2, and g and h of level 3. The root r has level 0. The height of the tree is 3.

National University of computer and emerging sciences

Foundation for Advancement of Science and Technology

Just Example. Not verified



Definition 3.16 Let T be a tree with root r. Then for any vertices x and y

- x is a ***descendant*** of y if y is on the unique path from x to r;
- x is a ***child*** of y if x is a descendant of y and exactly one level below y;
- x is an ***ancestor*** of y if x is on the unique path from y to r;
- x is a ***parent*** of y if x is an ancestor of y and exactly one level above y;
- x is a ***sibling*** of y if x and y have the same parent.

Definition 3.17 A tree in which every vertex has at most two children is called a ***binary tree***. If every parent has exactly two children we have a ***full binary tree***. Similarly, if every vertex has at most k children then the tree is called a ***k-nary tree***.

Example 3.7 Trees can be used to store information for quick access. Consider the following string of numbers:

$$4, 2, 7, 10, 1, 3, 5$$

We can form a tree by creating a vertex for each number in the list. As we move from one entry in the list to the next, we place an item below and to the left if it is less than the previously viewed vertex and below and to the right if it is greater. If we add the restriction that no vertex can have more than two edges coming down from it, then we are forming a binary tree.

4, 2, 7, 10, 1, 3, 5

For the string above, we start with a tree consisting of one vertex, labeled 4 (see T_1). The next item in the list is a 2, which is less than 4 and so its vertex is placed on the left and below the vertex for 4. The next item, 7, is larger than 4 and so its vertex is placed on the right and below the vertex for 4 (see T_2).

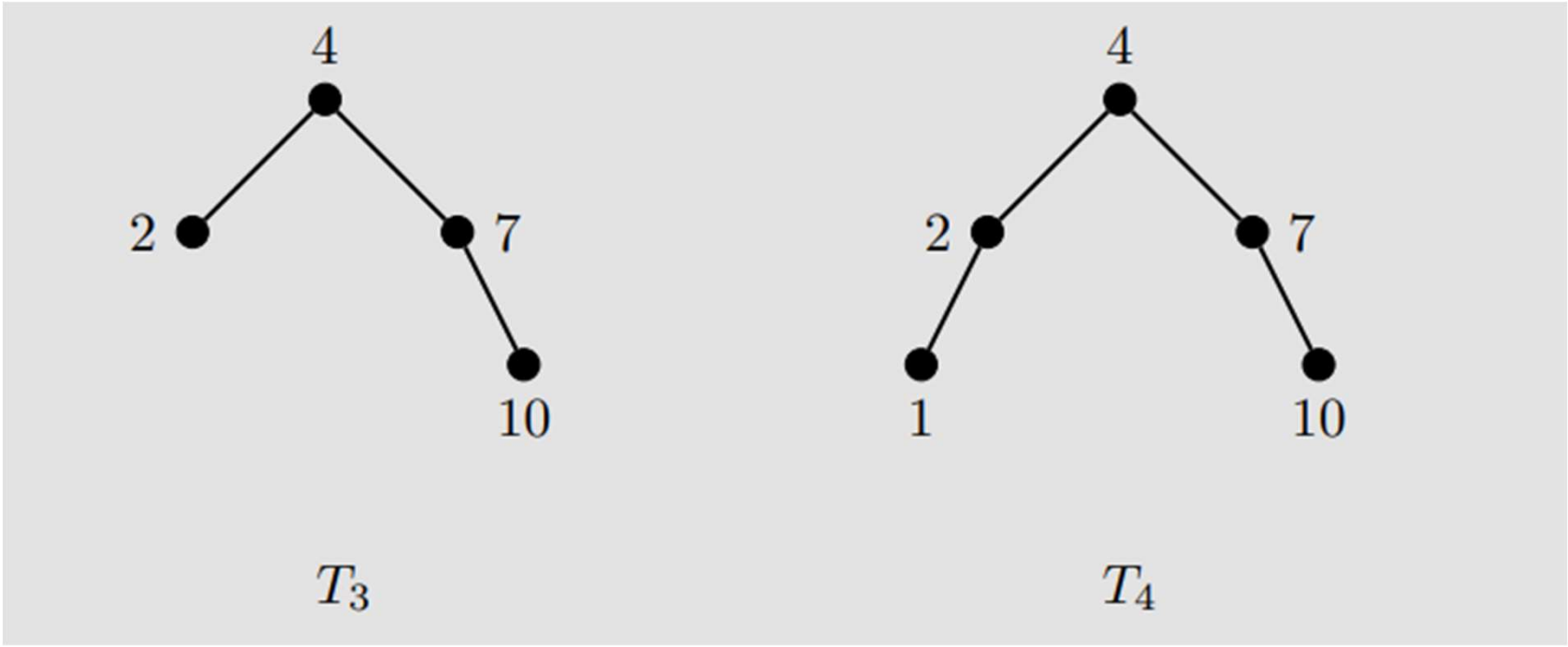
T_1

T_2

4, 2, 7, 10, 1, 3, 5

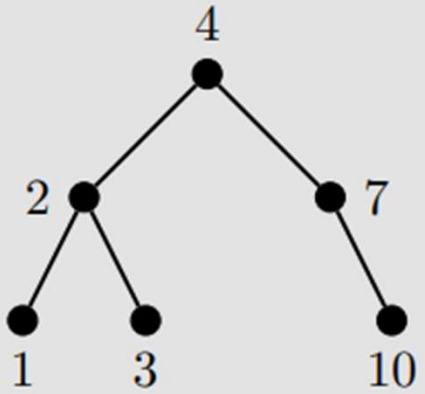
The next item in the list is 10. Since 4 already has two edges below it, we must attach the vertex for 10 to either 2 or 7. Since 10 is greater than 4, it must be placed to the right of 4 and since 10 is greater than 7, it must be placed to the right of 7 (see T_3). A similar reasoning places 1 to the left and below 2 (see T_4).

4, 2, 7, 10, 1, 3, 5

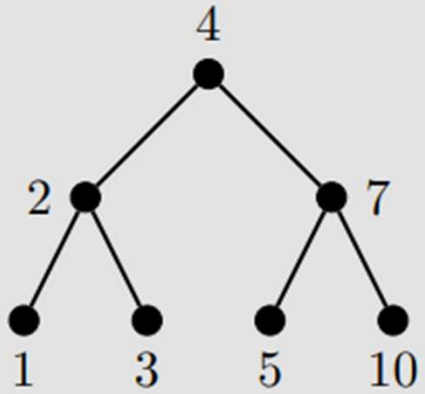


4, 2, 7, 10, 1, 3, 5

The next item is 3, which is less than 4 and so must be to the left of 4. Since 3 is greater than 2, it must be placed to the right of 2 (see T_5). The final item is 5, which is greater than 4 but less than 7, placing it to the right of 4 but to the left of 7 (see T_6).



T_5



T_6

Theorem 3.18 Let T be a binary tree with height h and l leaves. Then

(i) $l \leq 2^h$.

(ii) if T is a full binary tree and all leaves are at height h, then $l = 2^h$.

(iii) if T is a full binary tree, then $n = 2l - 1$.

Depth-First Search Tree

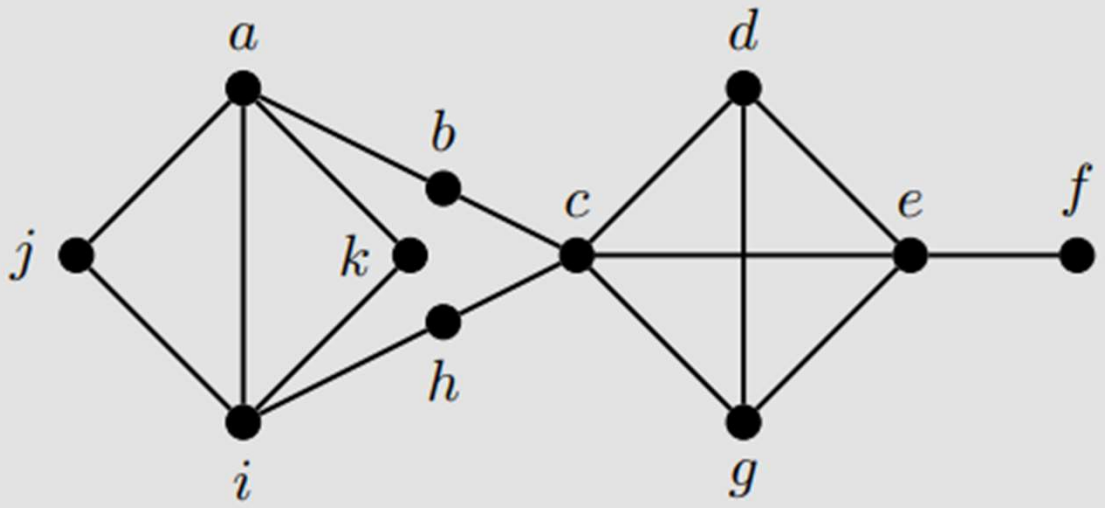
Input: Simple graph $G = (V, E)$ and a designated root vertex r.

Steps:

1. Choose the first neighbor x of r in G and add it to $T = (V, E')$.
2. Choose the first neighbor of x and add it to T. Continue in this fashion—picking the first neighbor of the previous vertex to create a path P. If P contains all the vertices of G, then P is the depth-first search tree. Otherwise continue to Step (3).
3. Backtrack along P until the first vertex is found that has neighbors not in T. Use this as the root and return to Step (1).

Output: Depth-first search tree T.

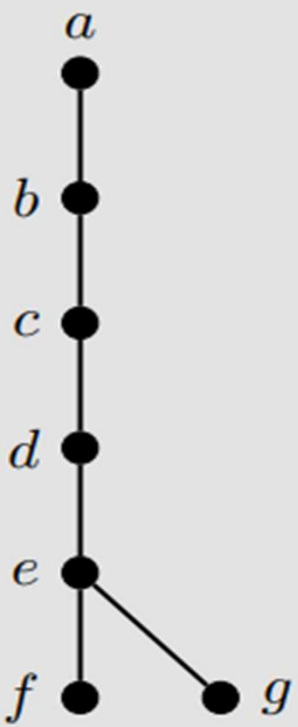
Example 3.8 Find the depth-first search tree for the graph below with the root a.



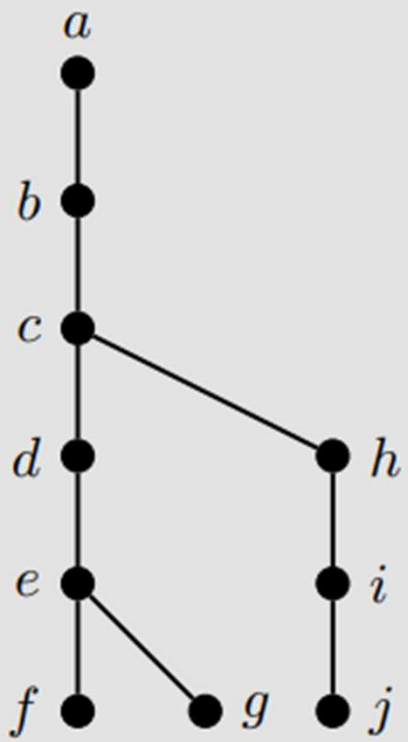
Solution:

Step 1: Since a is the root, we add b as it is the first neighbor of a. Continuing in this manner produces the path shown below. Note this path stops with f since f has no further neighbors in G.

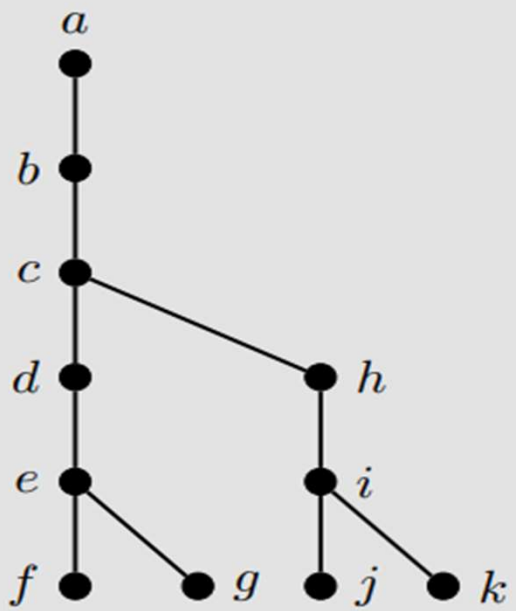
Step 2: Backtracking along the path above, the first vertex with an unchosen neighbor is e. This adds the edge eg to T. No other edges from g are added since the other neighbors of g are already part of the tree.



Step 3: Backtracking again along the path from Step 1, the next vertex with an unchosen neighbor is c. This adds the path $c\,h\,i\,j$ to T.



Step 4: Backtracking again along the path from Step 3, the next vertex with an unchosen neighbor is i. This adds the edge ik to T and completes the depth-first search tree as all the vertices of G are now included in T.



Output: The tree above is the depth-first search tree.

Breadth-First Search Tree

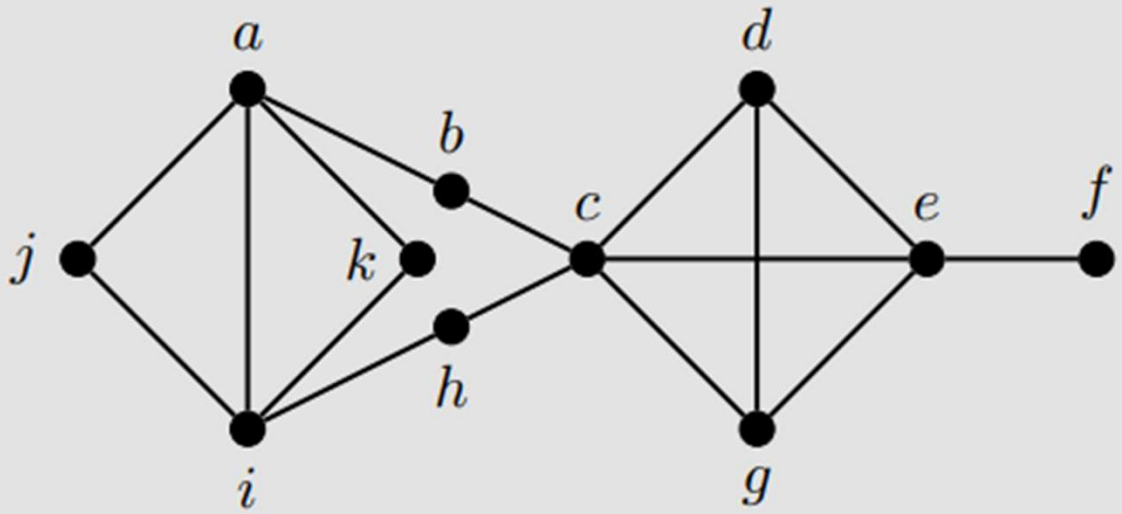
Input: Simple graph $G = (V, E)$ and a designated root vertex r.

Steps:

1. Add all the neighbors of r in G to $T = (V, E')$.
2. If T contains all the vertices of G, then we are done. Otherwise continue to Step (3).
3. Beginning with x, the first neighbor of r that has neighbors not in T, add all the neighbors of x to T. Repeat this for all the neighbors of r.
4. If T contains all the vertices of G, then we are done. Otherwise repeat Step (3) with the vertices just previously added to T.

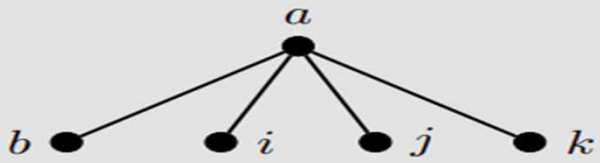
Output: Breadth-first tree T.

Example 3.9 Find the breadth-first search tree for the graph below with the root a.

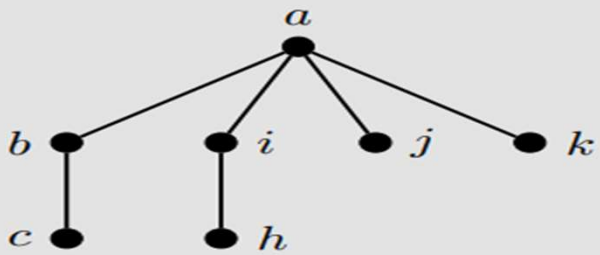


Solution:

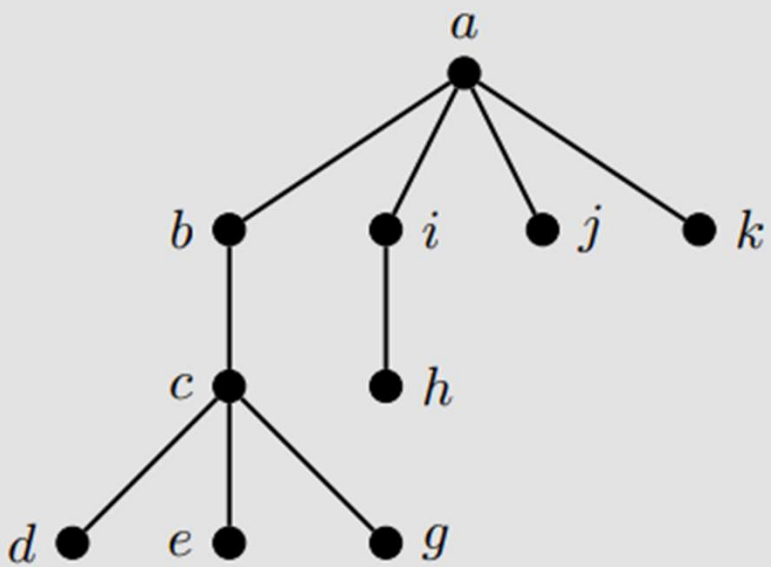
Step 1: Since a is the root, we add all of the neighbors of a to T.



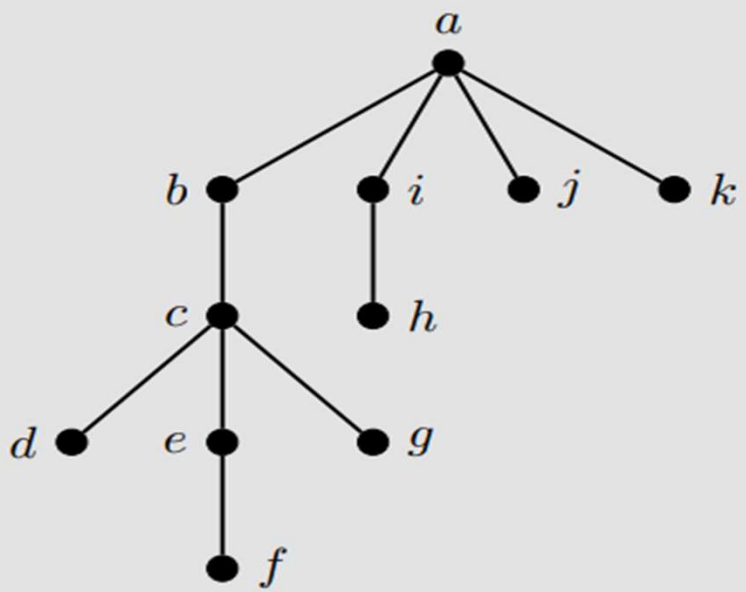
Step 2: We next add all the neighbors of b, i, j, and k, that are not already in T, beginning with b as it is the first neighbor of a that was added in Step 1. This adds the edge bc. Moving to i we add the edge ih. No other edges are added since j and k do not have any unchosen neighbors.



Step 3: We next add all the neighbors of c not in T, namely d, e, and g. No other vertices are added since all the neighbors of h are already part of the tree T.

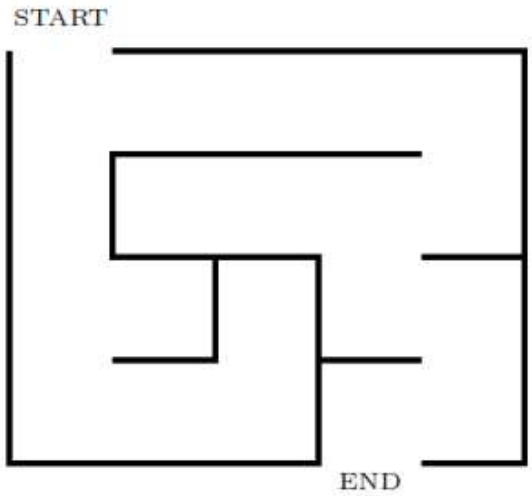


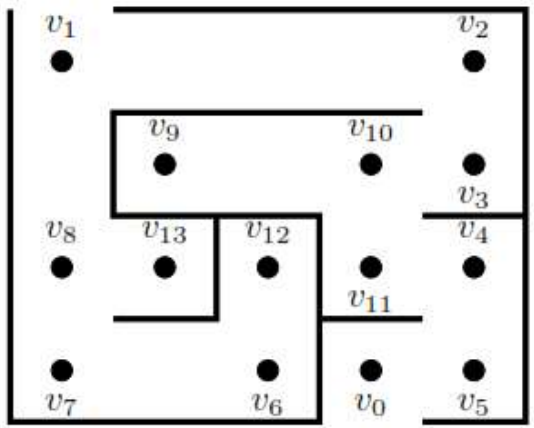
Step 4: Since d has no unchosen neighbors, we move ahead to adding the unchosen neighbors of e. This completes the breadth-first search tree as all the vertices of G are now included in T.

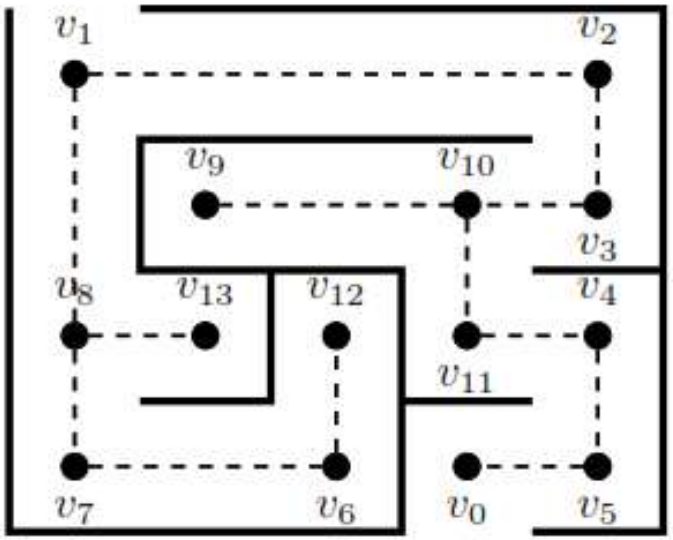


Output: The tree above is the breadth-first search tree.

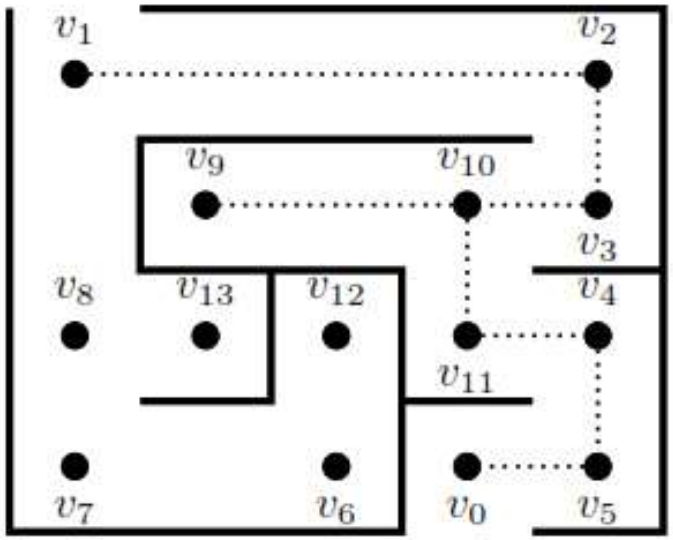
Mazes



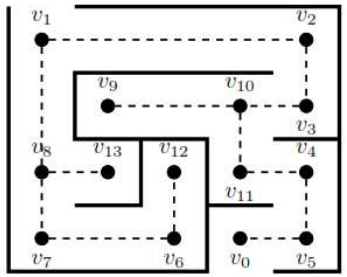




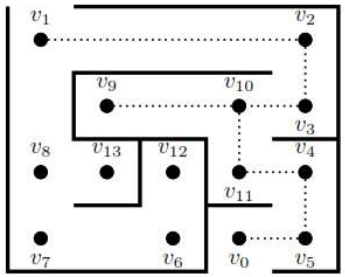
Breadth-First Search



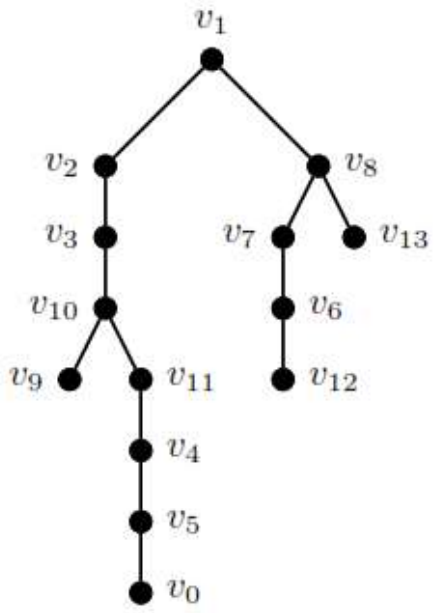
Depth-First Search



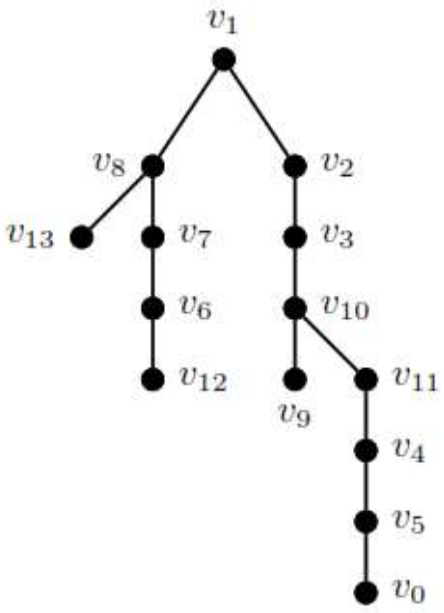
Breadth-First Search



Depth-First Search



Maze Breadth-First Tree



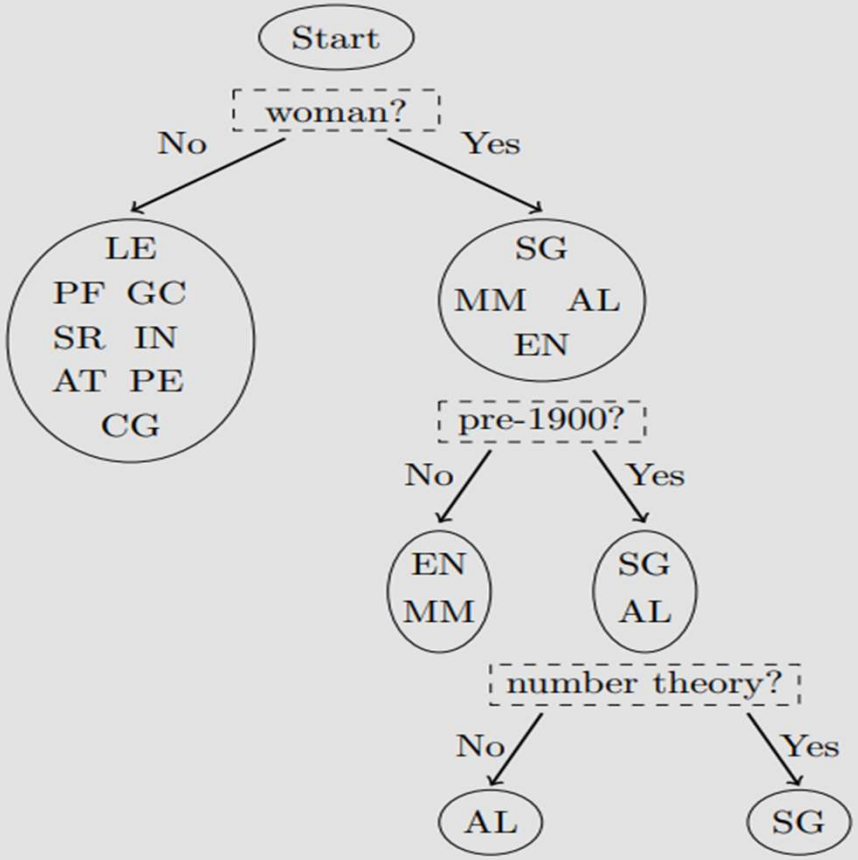
Maze Depth-First Tree

Decision Trees

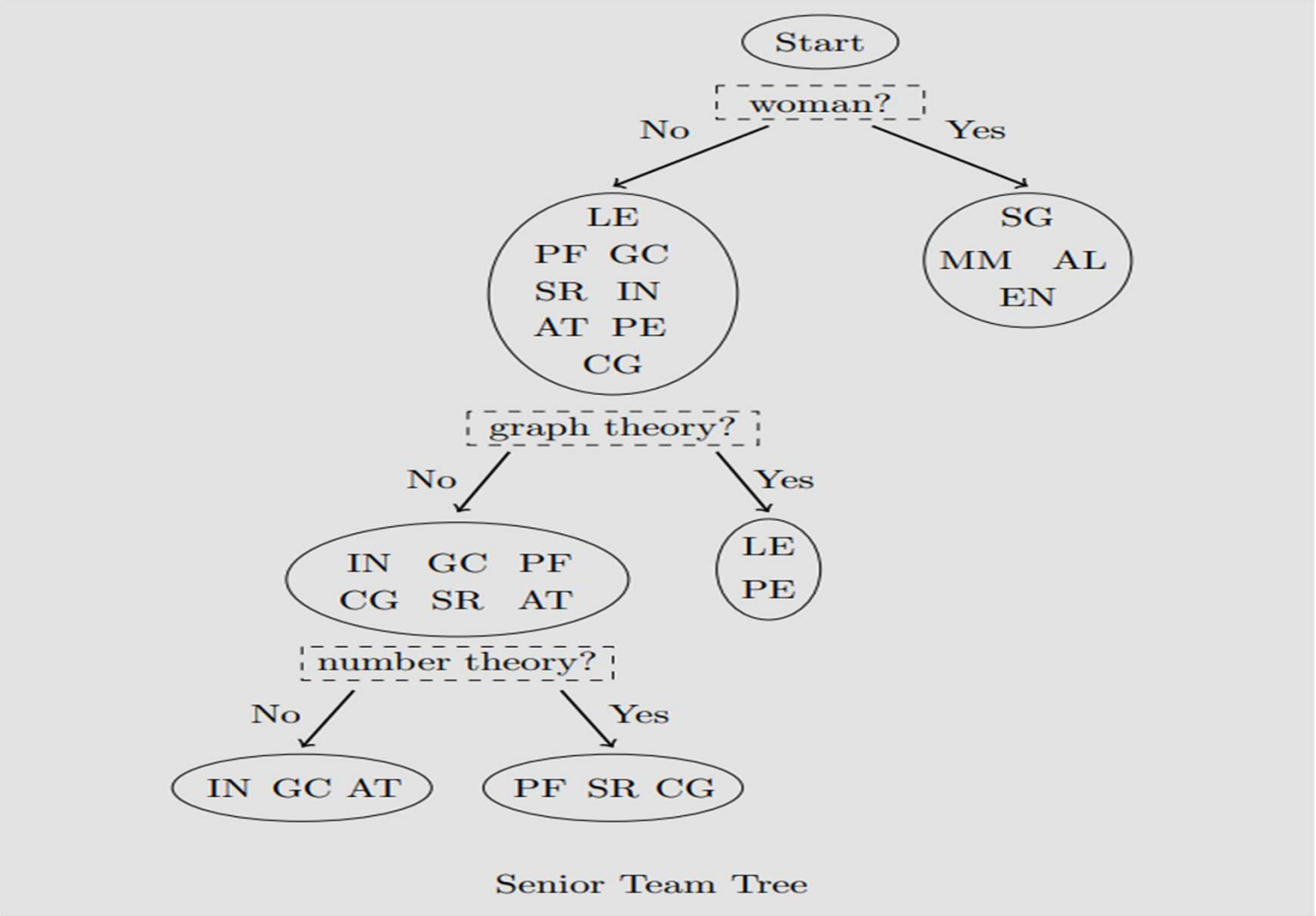
Example 3.11 The junior and senior math majors are facing off on a Mathematician Guess Who game. Below is a list of the cards. The junior team picked Pierre de Fermat and asked the questions (in order) "Is your person a woman? From pre-1900's? Known for Number Theory?" The senior team picked Sophie Germain and asked "Is your person a woman? Known for Graph Theory? Known for Number Theory?" Use a decision tree to determine which team won.

Pierre de Fermat (PF)	Carl Gauss (CG)	Leonhard Euler (LE)
Srinivasa Ramanujan (SR)	Alan Turing (AT)	Emmy Noether (EN)
Maryam Mirzakhani (MM)	Georg Cantor (GC)	Isaac Newton (IN)
Sophie Germain (SG)	Paul Erdös (PE)	Ada Lovelace (AL)

Solution: Below are two trees representing the sequence of questions asked by each team. Since only the junior team has a single person in their final set, we know they won the game after 3 questions.



Junior Team Tree



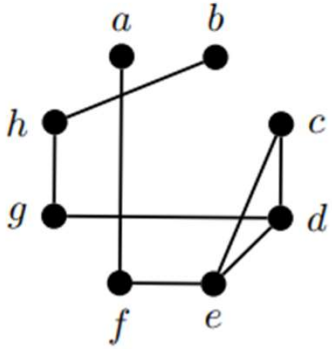
Senior Team Tree

https://ademos.people.uic.edu/Chapter24.html

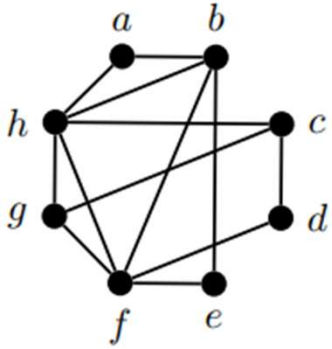
EX #: 3.5
Problems: 3.1-3.8, 3.13-3.17, 3.23

Connectivity Measures

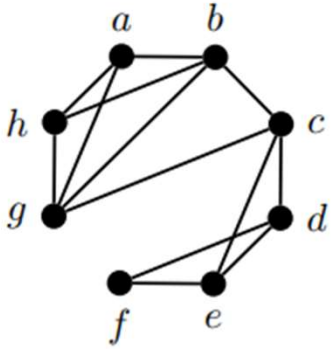
Definition 4.1 A ***cut-vertex*** of a graph G is a vertex v whose removal disconnects the graph, that is, G is connected but $G - v$ is not. A set S of vertices within a graph G is a ***cut-set*** if $G - S$ is disconnected.



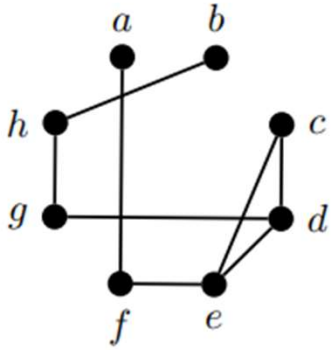
G_1



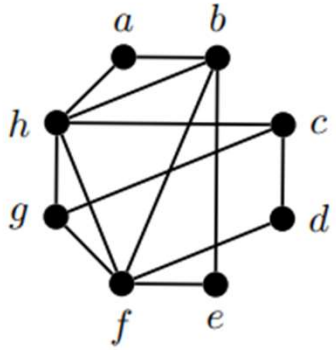
G_2



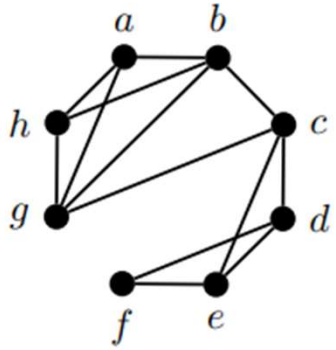
G_3



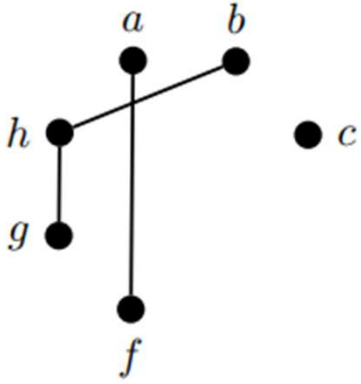
G_1



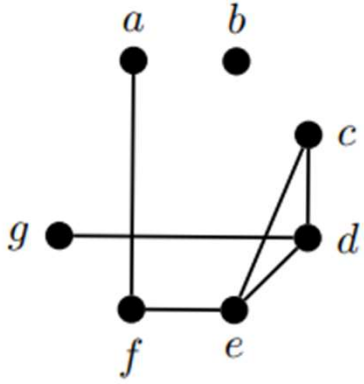
G_2



G_3



$G_1 - \{d, e\}$

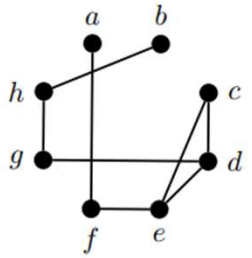


$G_1 - \{h\}$

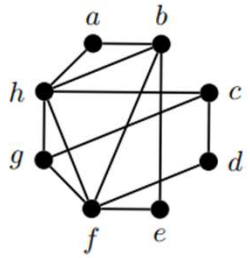
4.1.1 k-Connected

Definition 4.2 For any graph G, we say G is **k-connected** if the smallest cut-set is of size at least k.

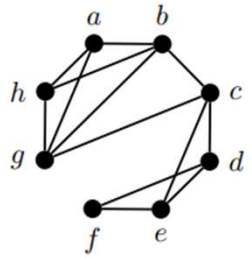
Define the connectivity of G, $\kappa(G) = k$, to be the maximum k such that G is k-connected, that is there is a cut-set S of size k, yet no cut-set exists of size $k-1$ or less. Define $\kappa(K_n) = n-1$.



G_1



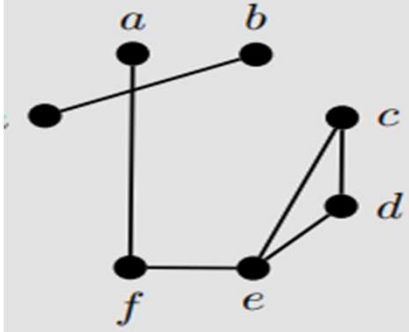
G_2



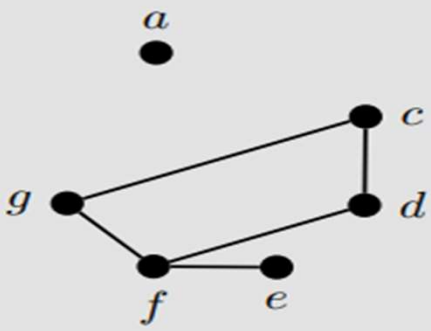
G_3

Example 4.1 Find $\kappa(G)$ for each of the graphs shown above on page 169.

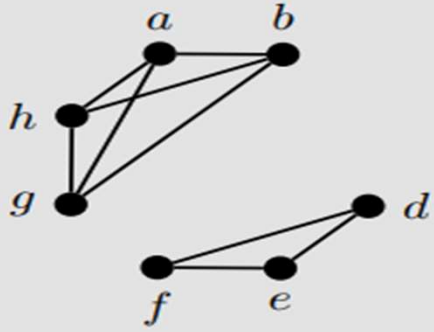
Solution: The removal of any one of d, e, f, g, or h in G_1 will disconnect the graph, so $\kappa(G_1) = 1$. Similarly, $G_3 - c$ has two components and so $\kappa(G_3) = 1$. However, $\kappa(G_2) = 2$ since the removal of any one vertex will not disconnect the graph, yet $S = \{b, h\}$ is a cut-set. Note this means G_2 is both 1-connected and 2-connected, but not 3-connected.



$G_1 - \{g\}$



$G_2 - \{b, h\}$



$G_3 - \{c\}$

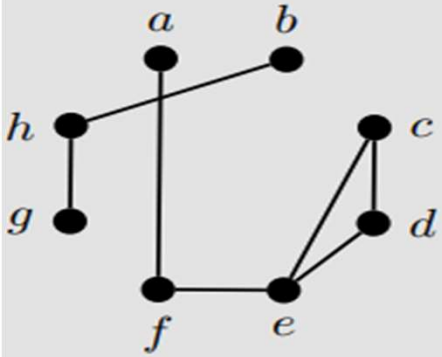
Definition 4.3 A ***bridge*** in a graph $G = (V, E)$ is an edge e whose removal disconnects the graph, that is, G is connected but $G - e$ is not. An ***edge-cut*** is a set $F \subseteq E$ so that $G - F$ is disconnected.

Definition 4.4 We say G is **k-edge-connected** if the smallest edge-cut is of size at least k.

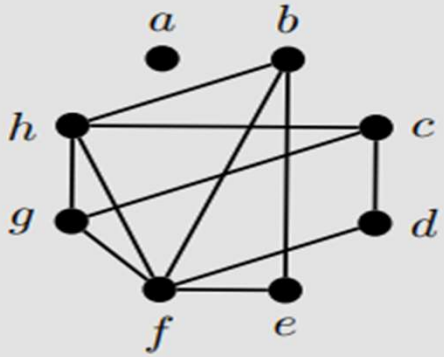
Define $\kappa'(G) = k$ to be the maximum k such that G is k-edge-connected, that is there exists a edge-cut F of size k, yet no edge-cut exists of size $k - 1$.

Example 4.2 Find $\kappa'(G)$ for each of the graphs shown on page 169.

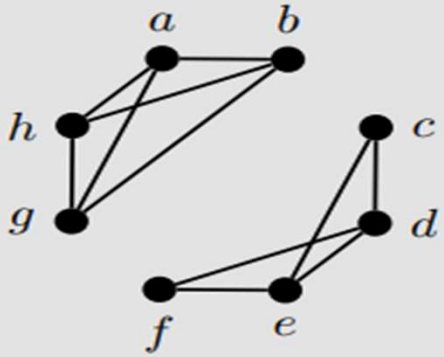
Solution: There are many options for a single edge whose removal will disconnect G_1 (for example af or dg). Thus $\kappa'(G_1) = 1$. For G_2, no one edge can disconnect the graph with its removal, yet removing both ab and ah will isolate a and so $\kappa(G_2) = 2$. Similarly $\kappa'(G_3) = 2$, since the removal of bc and cg will create two components, one with vertices a, b, g, h and the other with c, d, e, f.



$G_1 - dg$



$G_2 - \{ab, ah\}$



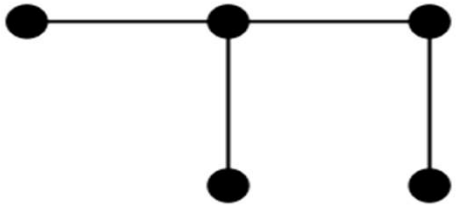
$G_3 - \{bc, cg\}$

Vertex-connectivity

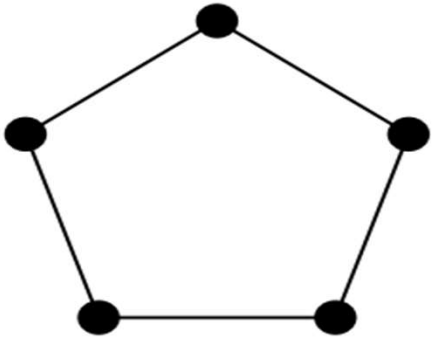
The **vertex-connectivity** (or simply the **connectivity**) of G, denoted by $\kappa(G)$, is defined by

$$\kappa(G) = \begin{cases} n-1, & \text{if } G \cong K_n; \\ \min\{|S| \mid S \text{ is a cut of } G\}, & \text{if } G \ncong K_n. \end{cases}$$

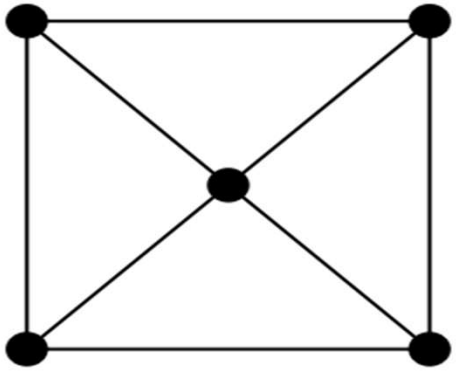
In other words, $\kappa(G)$ is the minimum number of vertices in G whose removal results in either a disconnected graph or a trivial graph (a singleton).



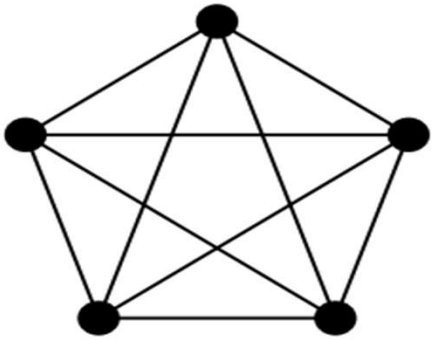
(a) Tree



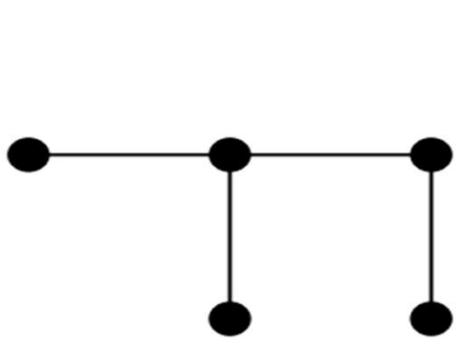
(b) Cycle C_5



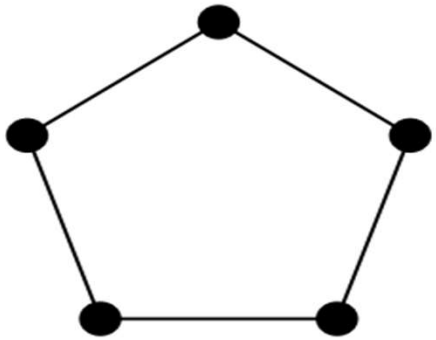
(c) Wheel W_5



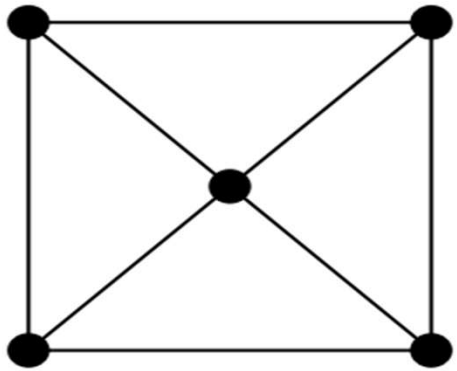
(d) Complete graph K_5



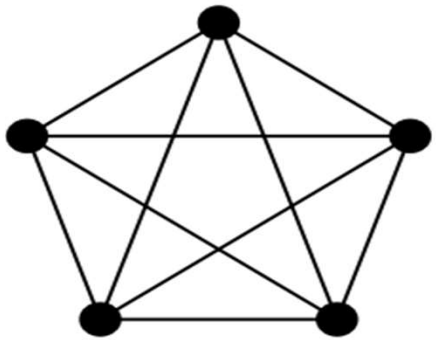
(a) Tree



(b) Cycle C_5



(c) Wheel W_5



(d) Complete graph K_5

(a) $\kappa(G) = 1$, (b) $\kappa(C_5) = 2$, (c) $\kappa(W_5) = 3$, and (d) $\kappa(K_5) = 4$.

(1) When we remove any r vertices $(1 \leq r \leq n-1)$ from the complete graph K_n, the resulting subgraph is never disconnected; it is a trivial graph if $r = n-1$. By definition, $\kappa(K_n) = n-1$.

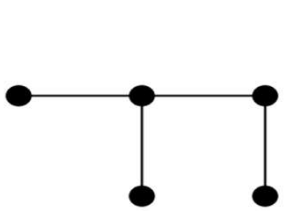
(2) We define, for convention, $\kappa(G) = 0$ if G is disconnected.

Edge-connectivity

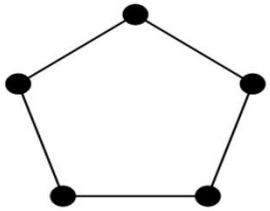
A subset F of E is called an **edge-cut** of G if $G - F$ is disconnected. Thus, an edge f in G is a bridge of G if and only if the singleton $\{f\}$ is an edge-cut of G. The **edge-connectivity** of G, denoted by $\kappa'(G)$, is defined by

$$\kappa'(G) = \begin{cases} 0, & \text{if } v(G) = 1; \\ \min\{|F| \mid F \text{ is an edge-cut of } G\}, & \text{if } v(G) \geq 2. \end{cases}$$

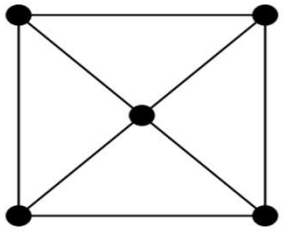
That is, $\kappa'(K_1) = 0$, and if $n \geq 2$, then $\kappa'(G)$ is the minimum number of edges in G whose removal results in a disconnected graph. Any edge-cut F of G with $|F| = \kappa'(G)$ is called a **minimum** edge-cut of G. Likewise, we define $\kappa'(G) = 0$ if G is disconnected.



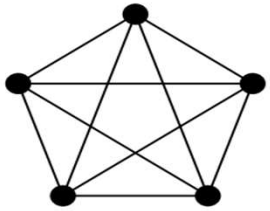
(a) Tree



(b) Cycle C_5



(c) Wheel W_5



(d) Complete graph K_5

(a) $\kappa'(G) = 1$;

(b) $\kappa'(C_5) = 2$;

(c) $\kappa'(W_5) = 3$;

(d) $\kappa'(K_5) = 4$.