Connectivity and Paths: Cuts and Connectivity

Dr. Rajiv Misra
Associate Professor
Dept. of Computer Science & Engg.
Indian Institute of Technology Patna
rajivm@iitp.ac.in

Preface

Recap of Previous Lecture:

In previous lecture, we have discussed Matchings in General Graphs *i.e.* Edmonds' Blossom Algorithm and also discuss the concepts of flower, stem and blossom.

Content of this Lecture:

In this lecture, we will discuss Connectivity and Paths *i.e.* Cuts and Connectivity

Connectivity of Graphs

Motivating Question

- How many vertices, or how many edges, can be deleted from a graph while keeping it connected?

Applications (Vertex Connectivity)

- Robustness of supercomputers to failures of processor nodes
- Sensor networks' resistance to individual sensor failure

Applications (Edge Connectivity)

- Robustness of supercomputers to failures of wires/fiber optics
- Reliability of road networks with road closures/accidents
- Communication networks' resistance to link failure

Cuts and Connectivity

- A **good communication network** is hard to disrupt. We want the graph (or digraph) of possible transmissions to remain connected even when some vertices or edges fail. When communication links are expensive, we want to achieve these goals with few edges.

- Loops are irrelevant for connection, so in this lecture we assume that our graphs and digraphs **have no loops**, especially when considering degree conditions.

Connectivity

How many vertices must be deleted to disconnect a graph?

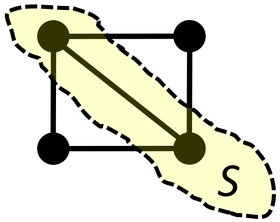
4.1.1. Definition. A **separating set** or **vertex cut** of a graph G is a set $S \subseteq V(G)$ such that $G–S$ has more than one component. The **connectivity** of G, written $\kappa(G)$, is the minimum size of a vertex set S such that $G–S$ is disconnected or has only one vertex. A graph G is ***k*-connected** if its connectivity is at least k.

A graph other than a complete graph is k-connected if and only if every separating set has size at least k. We can view **"*k*-connected"** as a structural condition, while **"connectivity *k*"** is the solution of an optimization problem.

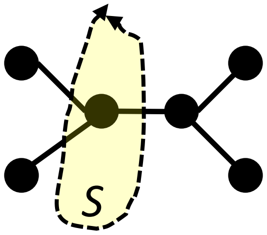
Example: Connectivity of K_n and $K_{m,n}$

- Because a clique has no separating set, we need to adopt a convention for its connectivity. This explains the phrase **"or has only one vertex"** in Definition 4.1.1. We obtain $\kappa(K_n) = n-1$, while $\kappa(G) \leq n(G)-2$ when G is not a complete graph. With this convention, most general results about connectivity remain valid on complete graphs.
- Consider a bipartition X, Y of $K_{m,n}$. Every induced subgraph that has at least one vertex from X and from Y is connected. Hence every separating set of $K_{m,n}$ contains X or Y. Since X and Y themselves are separating sets (or leave only one vertex), we have $\kappa(K_{m,n}) = \min\{m,n\}$. The connectivity of $K_{3,3}$ is 3; the graph is 1-connected, 2-connected, and 3-connected, but not 4-connected.

Examples: Vertex Connectivity

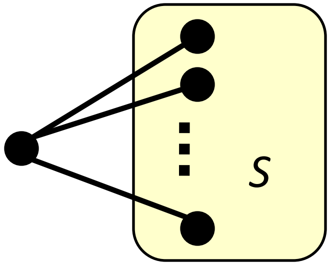


2-connected



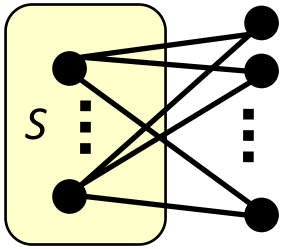
1-connected

K_n



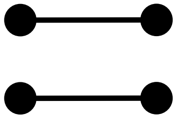
$(n\text{-}1)$-connected

$K_{m,n}$



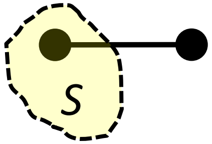
$\min(m,n)$-connected

0-connected

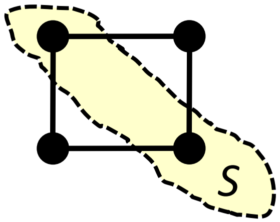


$2K_2$:
disconnected, so
0-connected

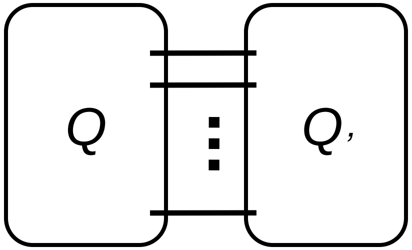
$k = 0$
0-connected
$\kappa = 0$



$k = 1$
1-connected
$\kappa = 1$



$k = 2$
2-connected
$\kappa = 2$



$k > 2$

$\kappa(Q_k) = k$

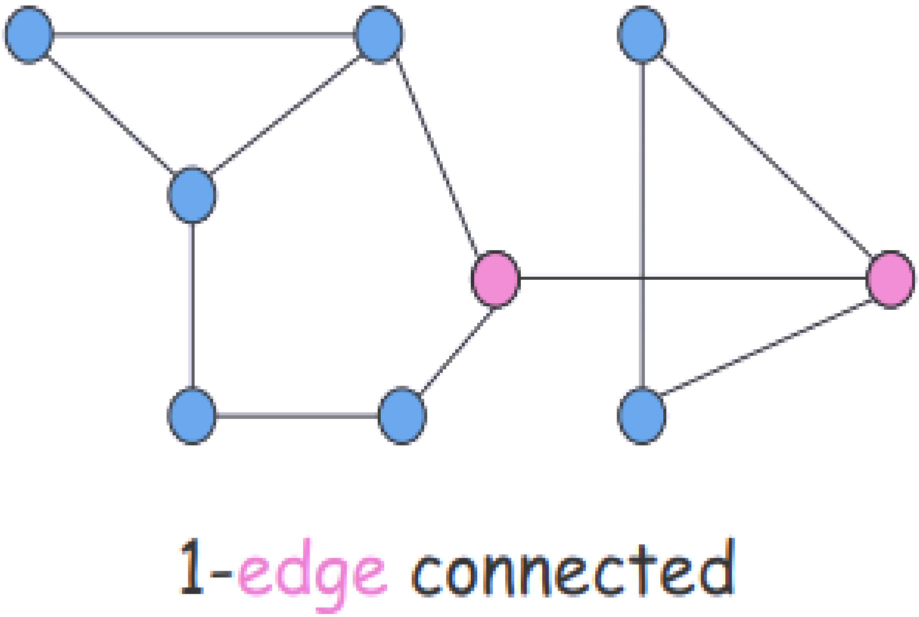
Examples: Vertex Connectivity

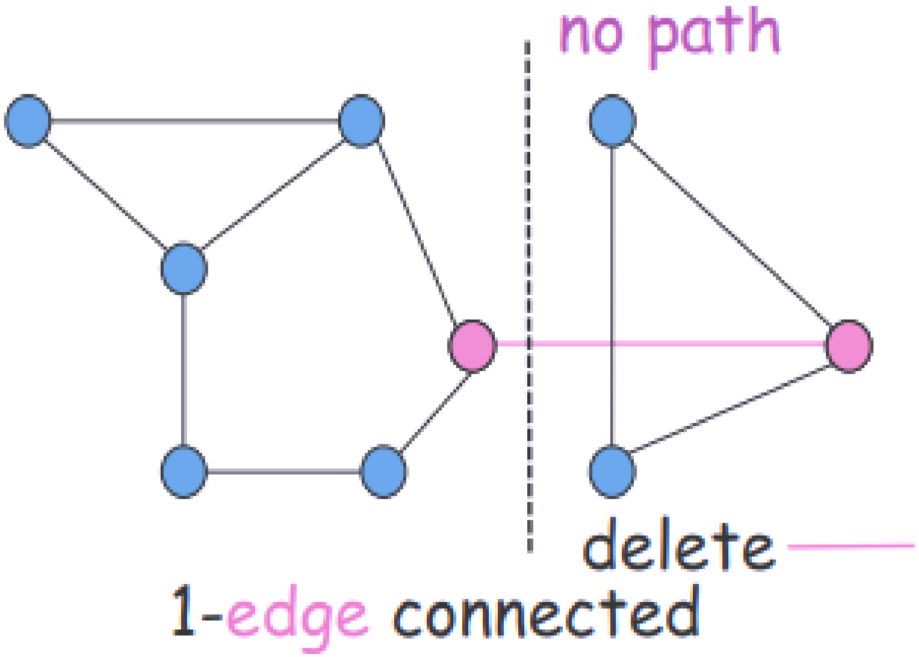
	K_1	K_2	K_3	K_4	K_n (n>3)	C_4	C_n (n>2)
Connectivity κ	0	1	2	3	n-1	2	2
1-connected?	N	Y	Y	Y	Y	Y	Y
2-connected?	N	N	Y	Y	Y	Y	Y
3-connected?	N	N	N	Y	Y	N	N

Edge Connectedness

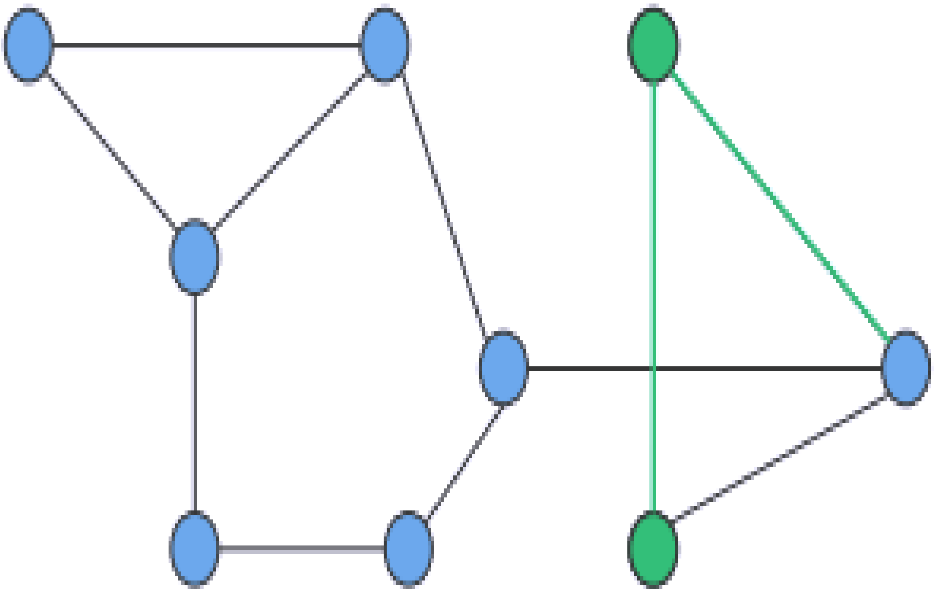
Definition: vertices *v*, *w* are ***k*-edge connected** if they remain connected whenever **fewer than** *k* edges are deleted.

Example: 1-edge connected

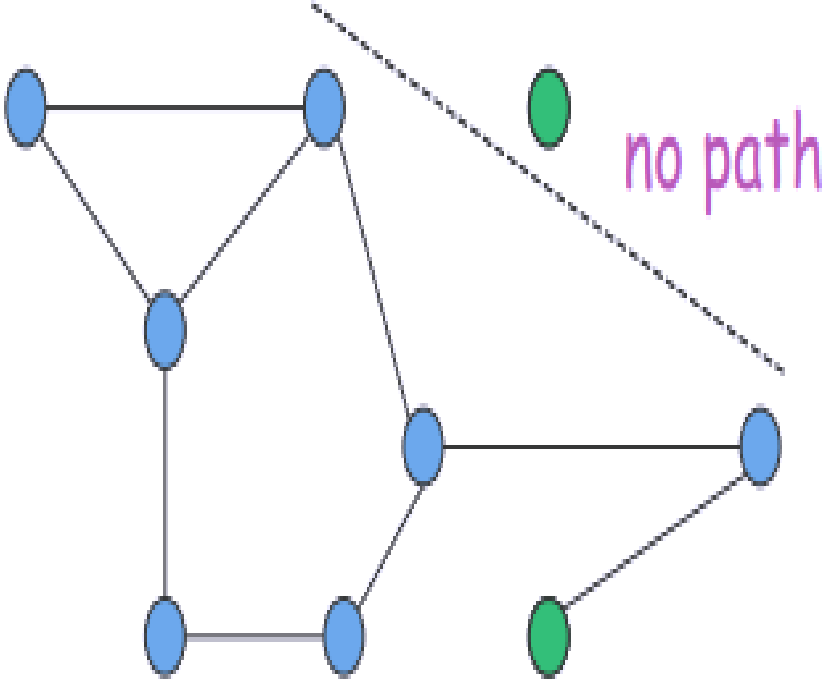




2-edge connected

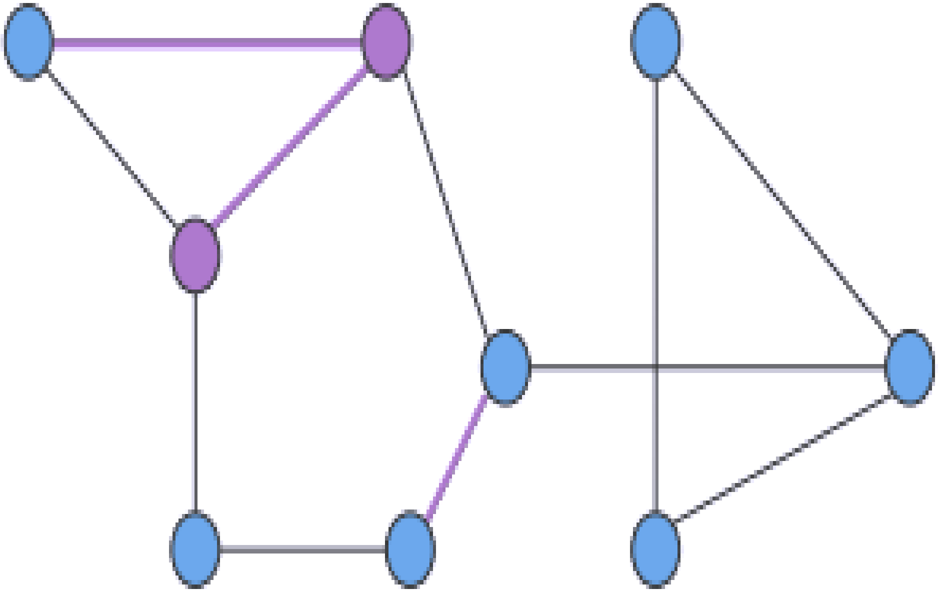


2-edge connected

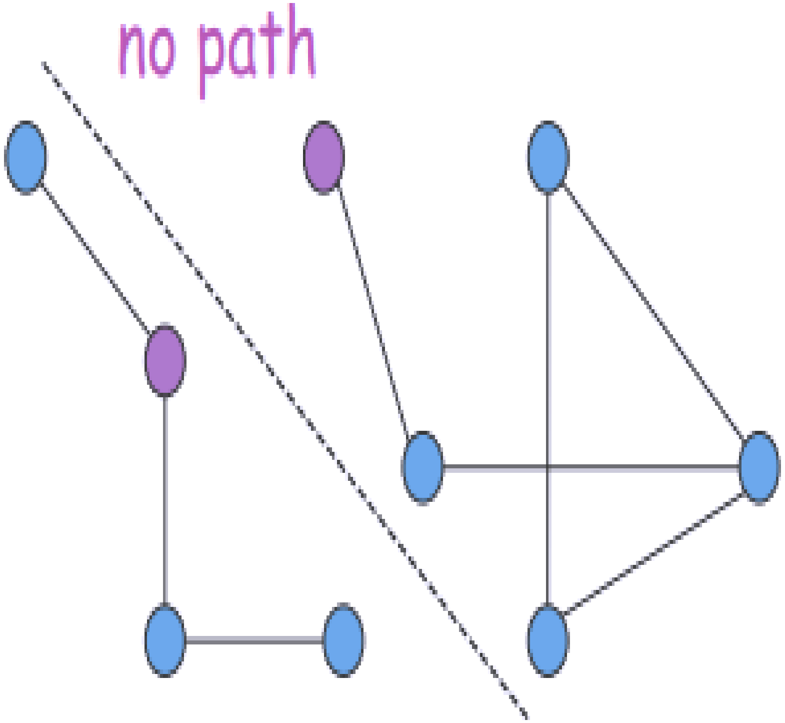


2-edge connected

3-edge connected



3-edge connected

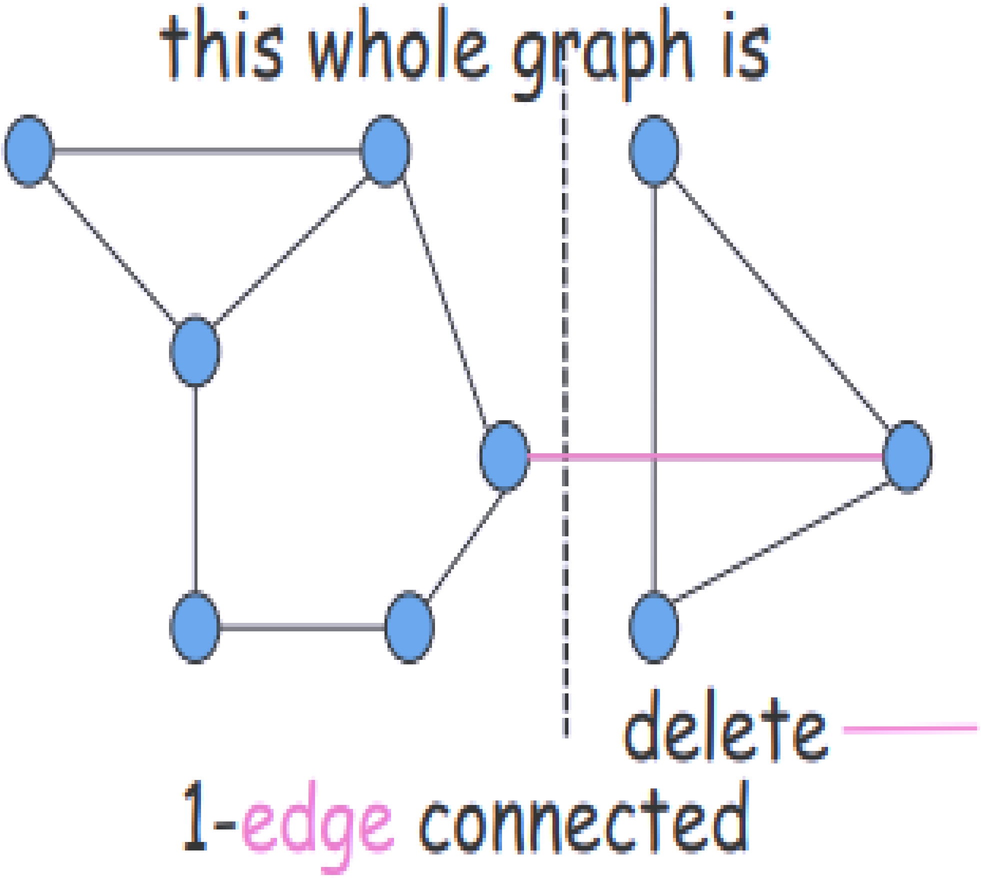


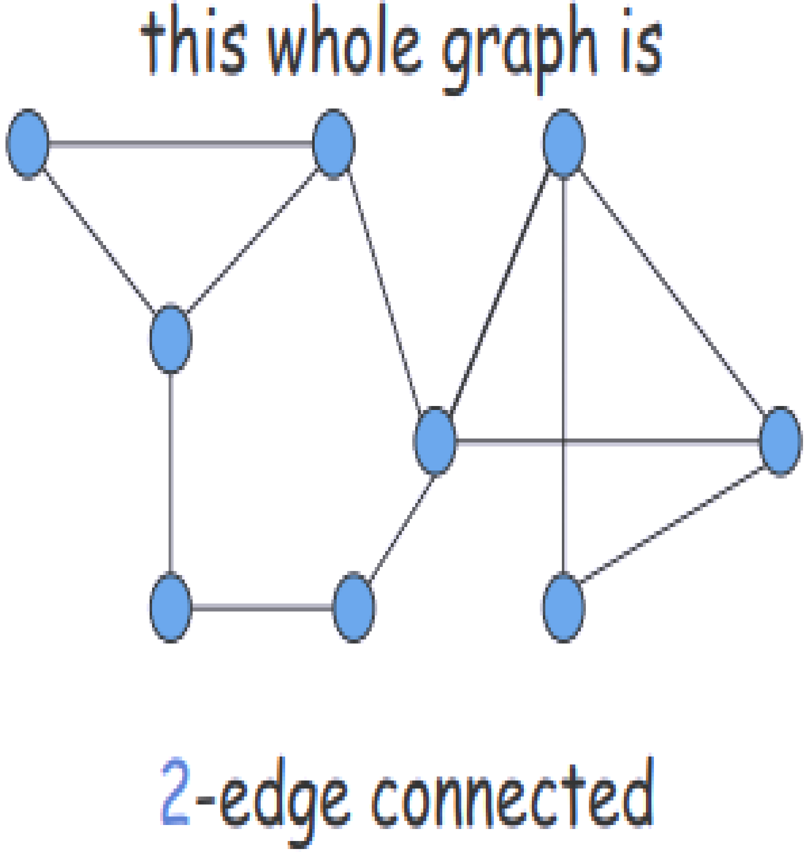
3-edge connected

k-edge Connectedness

- **Definition:** A graph is ***k*-edge connected** iff every two vertices are ***k*-edge connected**.
- **Connectivity** measures fault tolerance of a network: how many connections can fail without cutting off communication?

Examples: k-edge Connectedness



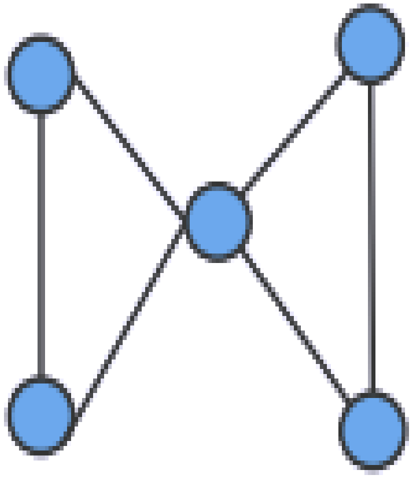


k-vertex Connectedness

- **k-vertex connectedness** defined similarly
- k-vertex connected

IMPLIES

k-edge connected not conversely:



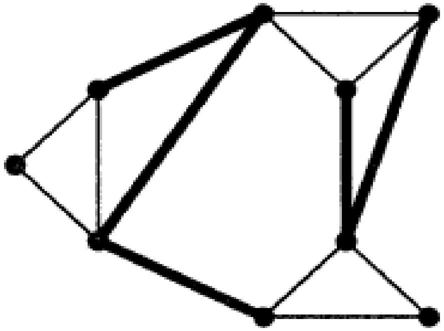
2-edge connected

1-vertex connected

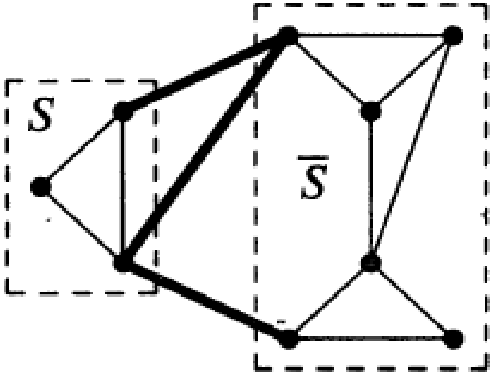
Definition: Edge-Connectivity 4.1.7

A **disconnecting set** of edges is a set $F \subseteq E(G)$ such that $G - F$ has more than one component. A graph is ***k*-edge-connected** if every disconnecting set has at least k edges. The **edge-connectivity** of G, written $\kappa'(G)$, is the minimum size of a disconnecting set (equivalently, the maximum k such that G is k-edge-connected).

Given $S,T \subseteq V(G)$, we write $[S,T]$ for the set of edges having one endpoint in S and the other in T. An **edge cut** is an edge set of the form $[S,\overline{S}]$ where S is a nonempty proper subset of $V(G)$ and $\overline{S}$ denotes V(G) - S.



disconnecting set



edge cut

Remark 4.1.8

Disconnecting set vs. edge cut

Every edge cut is a disconnecting set. Since $G - [S, \overline{S}]$ has no path from S to $\overline{S}$. The converse is false, since a disconnecting set can have extra edges.

Every minimal disconnecting set of edges is an edge cut (when $n(G) > 1$).

If G-F has more than one component for some $F \subseteq E(G)$, then for some component H of G-F we have deleted all edges with exactly one endpoint in H, Hence F contains the edge cut $[V(H), \overline{V(H)}]$, and F is not a minimal disconnecting set unless $F = [V(H), \overline{V(H)}]$

Connectivity and Min Degree for Simple Graphs

4.1.9 Theorem. (Whitney [1932a]) If G is a simple graph, then

$$\kappa(G) \le \kappa'(G) \le \delta(G).$$

Proof.

Proof of $\kappa'(G) \le \delta(G)$: The edges incident to a vertex of minimum degree are a disconnecting set.

Proof of $\kappa(G) \le \kappa'(G)$:

Let F be a minimum disconnecting set of G of size $\kappa'(G)$, which is therefore equal to an edge cut $[S, V(G)-S]$ by Remark 4.1.8.

Case 1 Every vertex of S is adjacent to every vertex of $V(G)-S$.

Then $\kappa'(G) = |[S,V(G)-S]| \ge n-1$, and $n-1 \ge \kappa(G)$ we already knew.

Case 2 There exist vertices $x \in S$ and $y \in V(G)-S$ with $xy \notin E(G)$.

Define $T = \big(N(x) \cap (V(G)-S) \big)$

$$\cup$$

$$\{z \in S-\{x\} : N(x) \cap (V(G)-S) \ne \emptyset\}.$$

Connectivity and Min Degree for Simple Graphs

4.1.9 Theorem. (Whitney [1932a]) If G is a simple graph, then

$$\kappa(G) \leq \kappa'(G) \leq \delta(G).$$

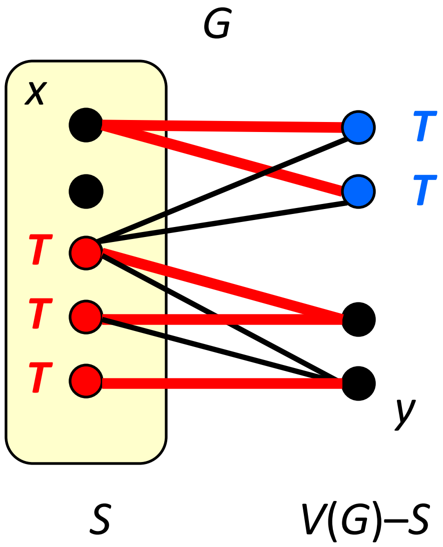
Proof. Proof of $\kappa(G) \leq \kappa'(G)$:

Case 2 There exist vertices $x \in S$ and $y \in V(G)-S$ with $xy \notin E(G)$.

Define $T = (\, N(x) \cap (V(G)-S)\,) \cup \{z \in S-\{x\} : N(x) \cap (V(G)-S) \neq \emptyset\}$.

- T is a vertex cut because all x,y-paths would would have to cross through T.
- The edges F_T both incident to T and in the edge cut $[S,V(G)-S]$ are a disconnecting set.
- Every vertex of T has at least one neighbor, so $|[S,V(G)-S]| \geq |F_T| \geq |T|$.

We have found a vertex cut T with size at most the size of a minimum edge cut $[S,V(G)-S]$, and therefore $\kappa(G) \leq \kappa'(G)$.



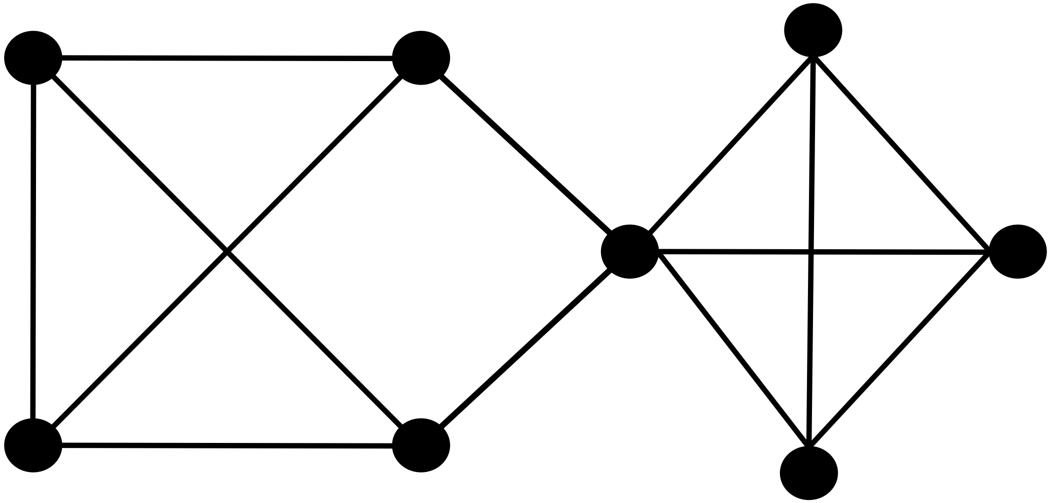
(|T| bold edges)

Example: Possibility of $\kappa < \kappa' < \delta$ 4.1.10

For graph G below,

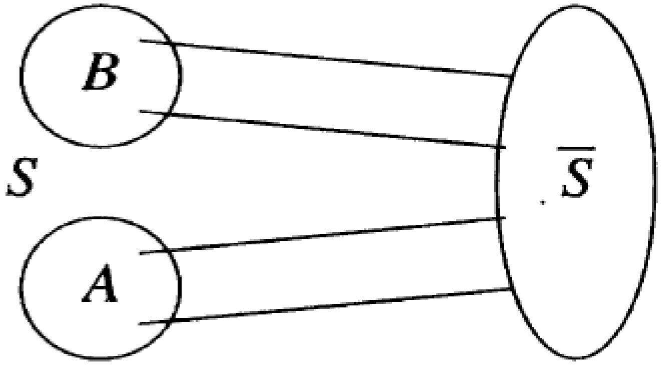
$\kappa(G) = 1$, $\kappa'(G) = 2$, and $\delta(G) = 3$. Note that no minimum edge cut isolates a vertex.

Each inequality can be arbitrarily weak, When $G = K_m + K_m$, we have $\kappa(G)=\kappa'(G) = 0$ but $\delta(G) = m$-1. When G consists of two m-cliques sharing a single vertex, we have $\kappa'(G)= \delta(G) = m$-1 but $\kappa(G)= 1$.



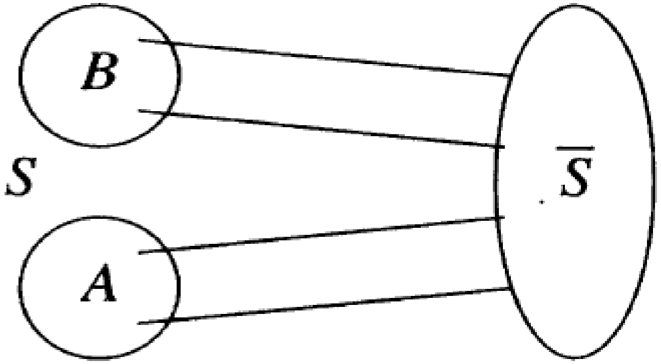
Definition: Bond

- A **bond** is a minimal nonempty edge cut.
- Here **“minimal”** means that no proper nonempty subset is also an edge cut. We characterize bonds in connected graphs.



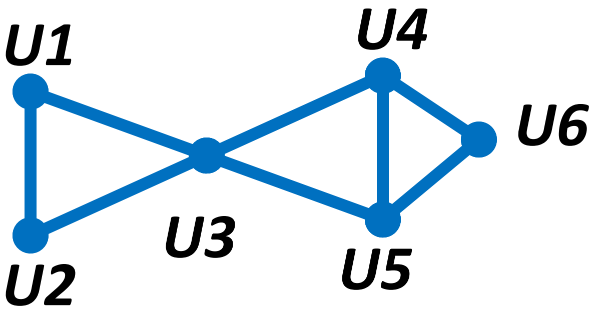
Proposition. If *G* is a connected graph, then an edge cut *F* is a bond if and only if *G-F* has exactly two components. 4.1.15

- **Proof:** Let F=[S, $\overline{S}$] be an edge cut. Suppose first that G-F has exactly two components, and let F’ be a proper subset of F, The graph G-F’ contains the two components of G-F plus at least one edge between them, making it connected, Hence F is a minimal edge cut and is a bond.
- For the converse, suppose that G-F has more than two components. Since G-F is the disjoint union of G[S] and G[$\overline{S}$], one of these has at least two components. Assume by symmetry that it is G[S]. We can thus write S=A U B, where no edges join A and B. Now the edge cuts [A,$\overline{A}$] and [B, $\overline{B}$] are proper subsets of F, so F is not a bond.

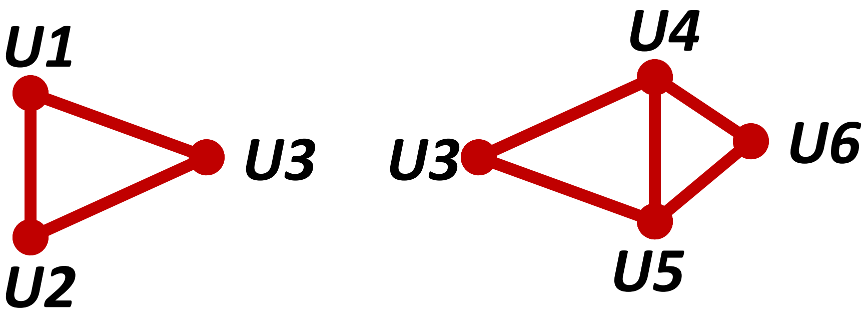


Block

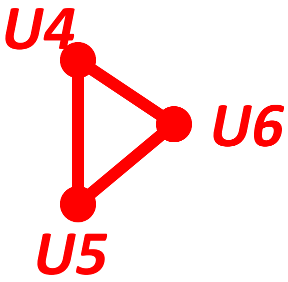
- A **Block** is a maximal nonseparable subgraph.



Blocks:



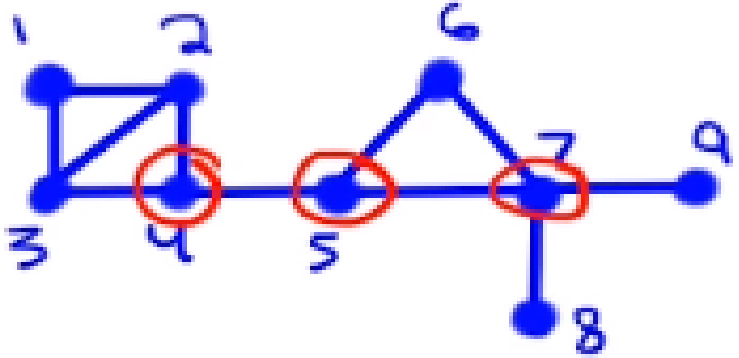
Notice that



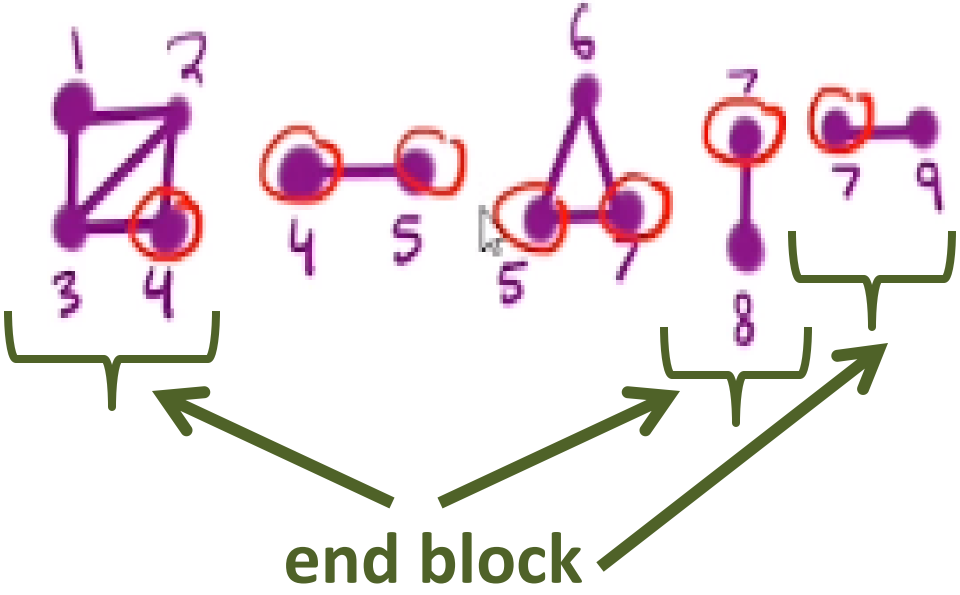
is nonseparable but not maximal nonseparable.

Theorem

- If G is a graph with at least one cut-vertex, then at least 2 of the blocks of G contain exactly 1 cut-vertex. These are called **end-blocks**



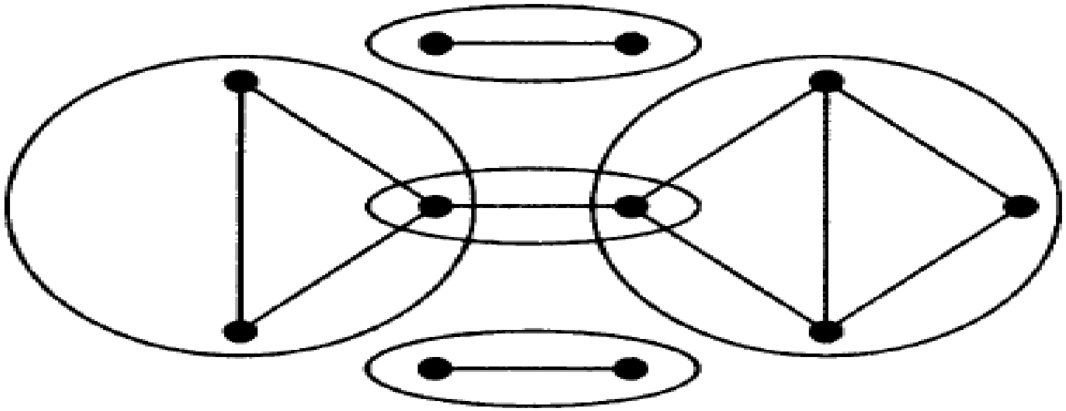
Blocks:



end block

Definition: Block 4.1.16

- A **block** of a graph *G* is a maximal connected subgraph of *G* that has no cut vertex. If G itself is connected and has no cut-vertex, then G is a block.
- A **block** is a maximal subgraph that cannot be disconnected by removing one vertex.
- **Example:** If H is a block of G, then H as a graph has no cut-vertex, but H may contain vertices that are cut-vertices of G, For example, the graph drawn below has five blocks; three copies of K_2, one of K_3, and one subgraph that is neither a cycle nor a complete graph.



Remark 4.1.18

Properties of Blocks:

- An edge of a cycle cannot itself be a block, since it is in a larger subgraph with no cut-vertex. Hence and edge is a block if and only if it is a cut-edge; the blocks of a tree are its edges.

- If a block has more than two vertices, then it is 2-connected. The blocks of a loopless graph are its isolated vertices, its cut-edges, and its maximal 2-connected subgraphs.

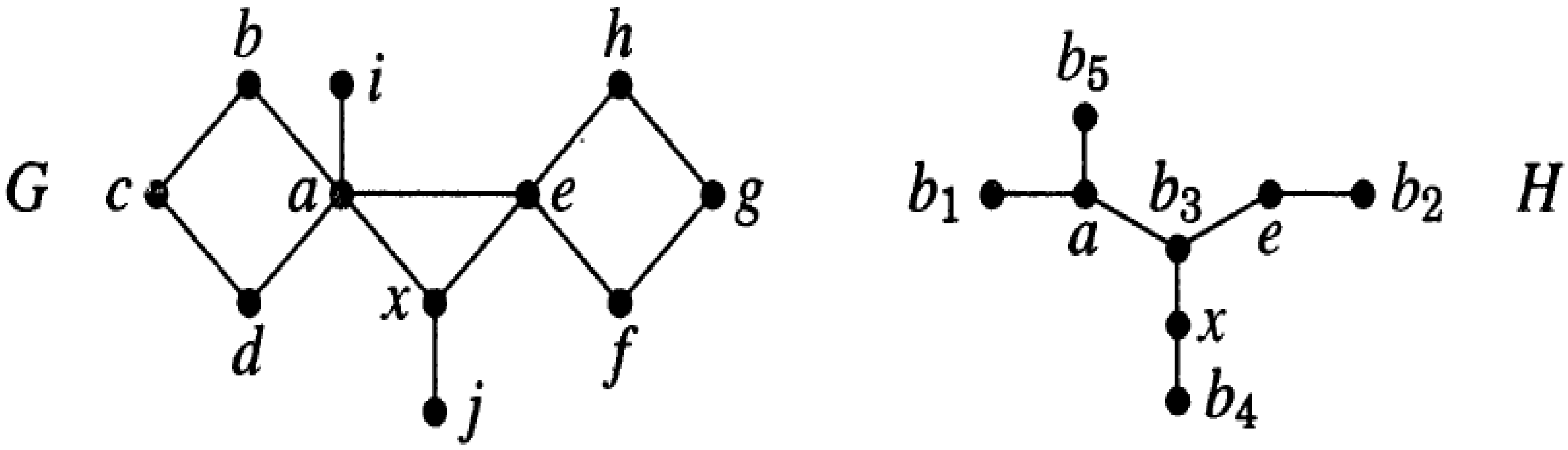
Proposition: Two blocks in a graph share at most one vertex. 4.1.19

- **Proof:** We use contradiction, Suppose that blocks B_1, B_2 have at least two common vertices, We show that B_1 U B_2 is a connected subgraph with no cut-vertex, which contradicts the maximality of B_1 and B_2.

- When we delete one vertex from B_i, what remains is connected, Hence we retain a path in B_i, from every vertex that remains to every vertex of $V(B_1) \cap V(B_2)$ that remains. Since the blocks have at least two common vertices, deleting a single vertex leaves a vertex in the intersection, We retain paths from all vertices to that vertex, so B_1 U B_2 cannot be disconnected by deleting one vertex.

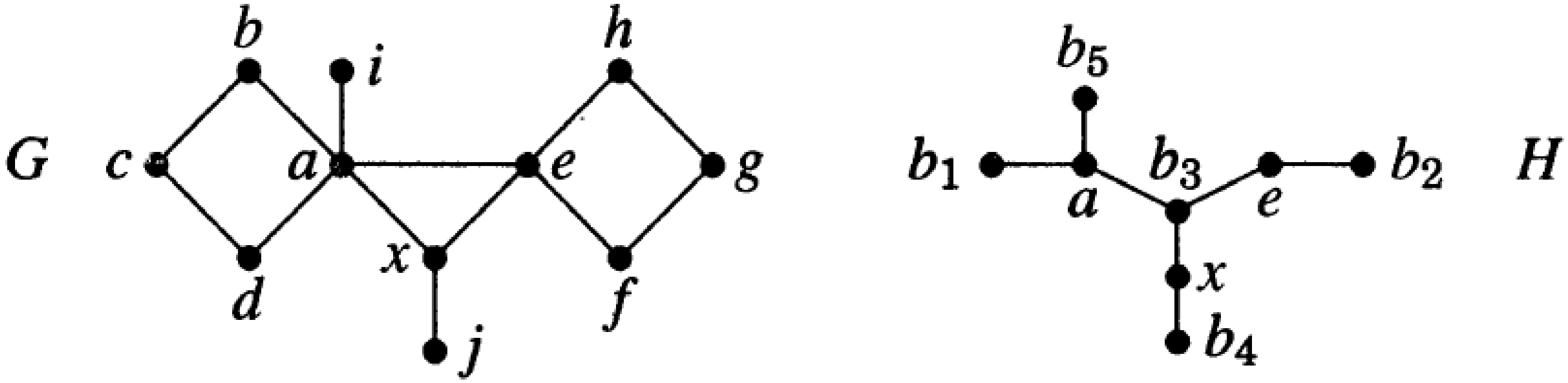
Definition: Block-cutpoint graph 4.1.20

- The **block-cutpoint graph** of a graph G is a bipartite graph H in which one partite set consists of the cut-vertices of G, and the other has a vertex b_i for each block B_i of G. We include vb_i as an edge of H if and only if $v \in B_i$.



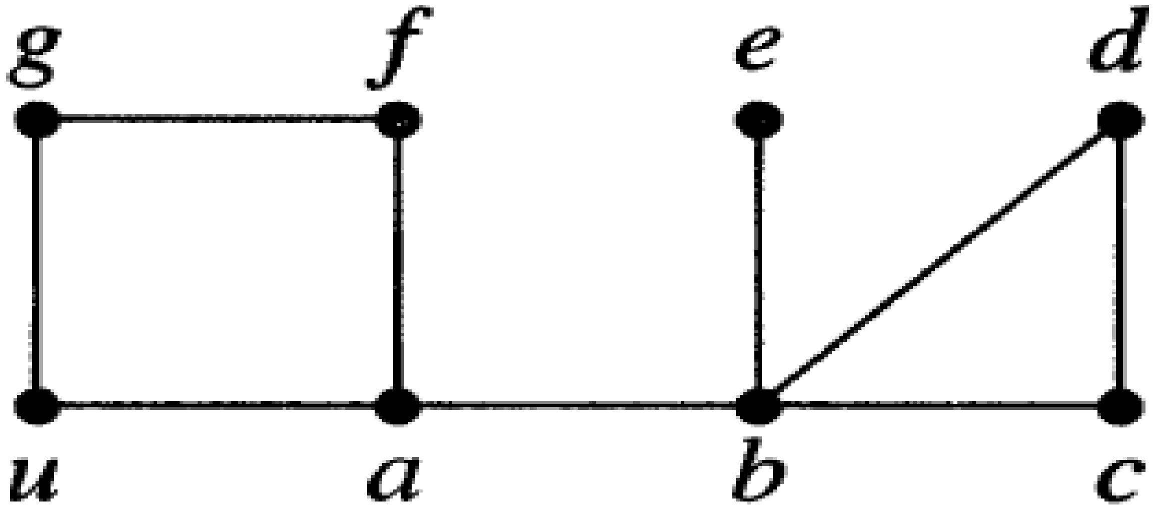
Contd...

- When G is connected, its block-cutpoint graph is a tree whose leaves are blocks of G, Thus a graph G that is not a single block has at least two blocks (**leaf blocks**) that each contain exactly one cut-vertex of G.
- Blocks can be found using a technique for searching graphs. In **Depth-First Search (DFS),** we explore always from the most recently discovered vertex that has unexplored edges (also called **backtracking**). In contrast, Breadth-First Search explores from the oldest vertex, so the difference between DFS and BFS is that in DFS we maintain the list of vertices to be searched as a Last-In-First-Out "stack" rather than a queue.



Example: Depth-First Search

- In the graph below, one depth-first search from *u* finds the vertices in the order *u, a, b, c, d, e, f, g*. For both BFS and DFS, the order of discovery depends on the order of exploring edges from a searched vertex.



Lemma 4.1.22

- If *T* is a spanning tree of a connected graph *G* grown by DFS from *u*, then every edge of *G* not in *T* consists of two vertices *v, w* such that *v* lies on the path *u*, *w*-path in *T*.

Proof:

- Let *vw* be an edge of *G*, with *v* encountered before *w* in the depth-first-search. Because *vw* be an edge, we cannot finish *v* before *w* is added to *T,* Hence *w* appears somewhere in the subtree formed before finishing *v*, and the path from *w* to *u* contains *v*.

Algorithm: Computing the block of a graph 4.1.23

Input: A connected graph G

Idea: Build a depth-first search tree T of G, discarding portions of T as blocks are identified, Maintain one vertex called ACTIVE.

Initialization: Pick a root x ϵ V(H); make x ACTIVE; set T={x}.

Iteration: Let v denote the current active vertex.

1) If v has an unexplored incident edge vw, then

   1A) If w $\notin$V(T), then add vw to T, mark vw explored, make w ACTIVE.

   1B) If w ϵ V(T), then w is an ancestor of v; mark vw explored.

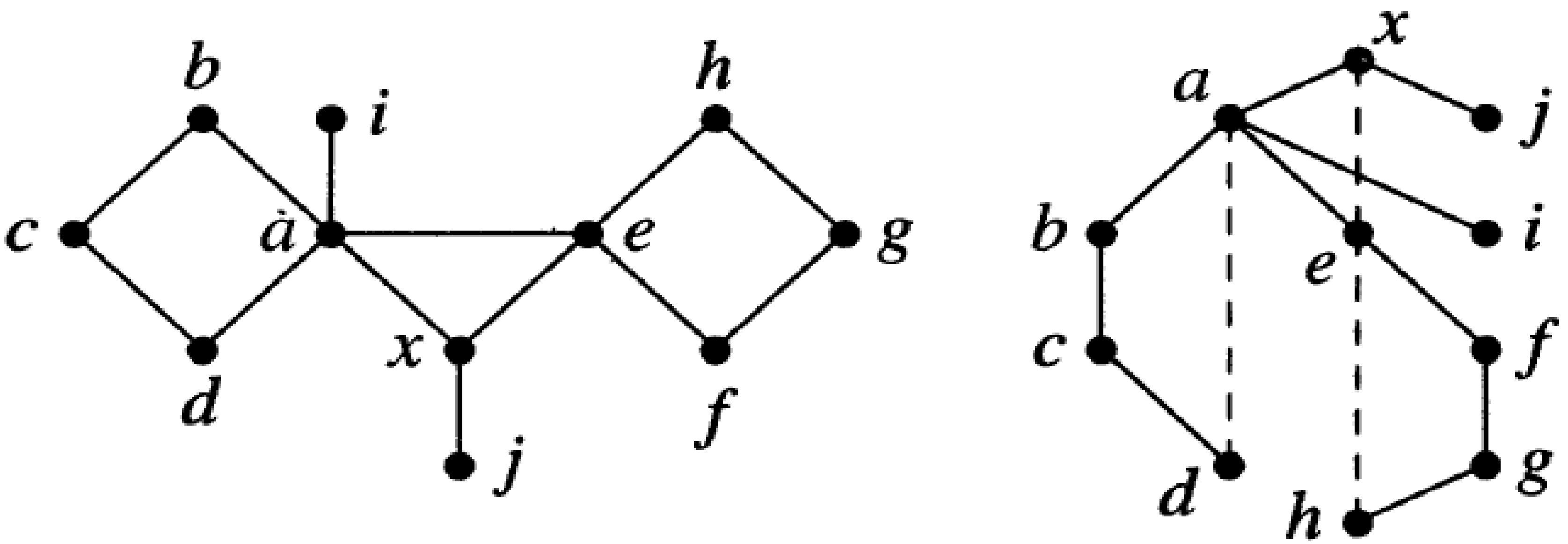
2) If v has no more unexplored incident edges, then

   2A) If v ≠ x, and w is the parent of v, make w ACTIVE, If no vertex in the current subtree T' rooted at v has an explored edge to an ancestor above w, then V(T') U {w} is the vertex set of a block; record this information and delete V(T') from T.

   2B) If v=x, terminate.

Example: Finding blocks

- For the graph below, one depth-first traversal from x visits the other vertices in the order *a, b, c, d, e, f, g, h, i, j*. We find blocks in the order *{a,b,c,d,}, {e, f, g, h}, {a, i}, {x, a, e}, {x, j}*. After finding each block, we deleted the vertices other than the highest.



Conclusion

- In this lecture, we have discussed **Cuts and Connectivity** *i.e.* vertex connectivity, edge connectivity, bond, blocks and also discuss the theorems based on the cuts and connectivity.

- In upcoming lecture, we will discuss the ***k*-Connected Graphs**.