Chap 3 Trees

Yih-Lang Li (李毅郎)
Computer Science Department
National Yang-Ming Chiao-Tung University, Taiwan

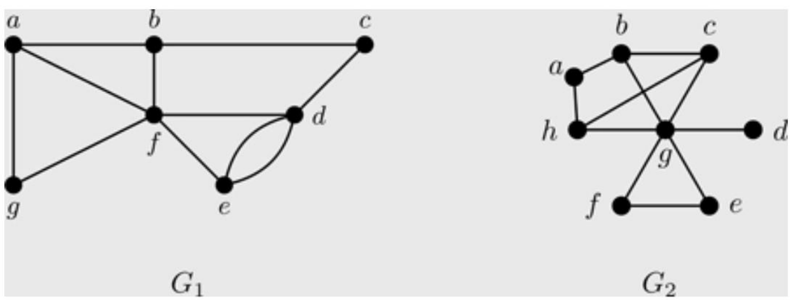
The sources of most figure images are from the textbook

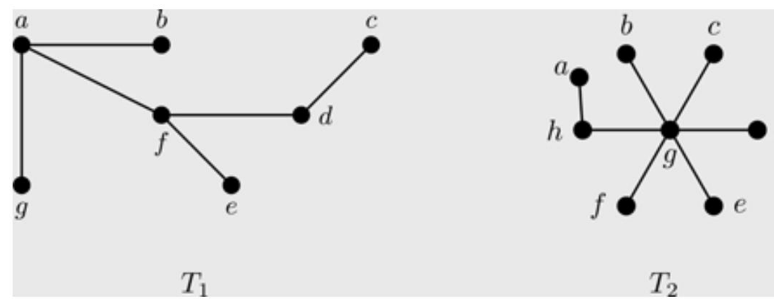
Outline

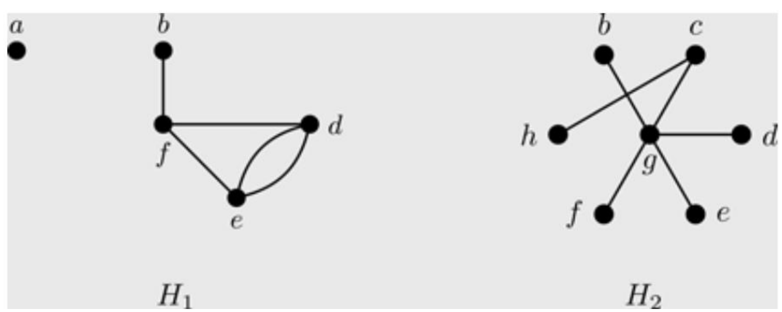
- Spanning Trees
- Tree Properties
- Rooted Trees
- Additional Applications

3.1 Spanning Trees

- **Definition 3.1** A graph G is
  - *acyclic* if there are no cycles or circuits in the graph.
  - a *tree* if it is both acyclic and connected.
  - a *forest* if it is an acyclic graph.
  - In addition, a vertex of degree 1 is called a *leaf*.
- **Definition 3.2** A *spanning tree* is a spanning subgraph that is also a tree.





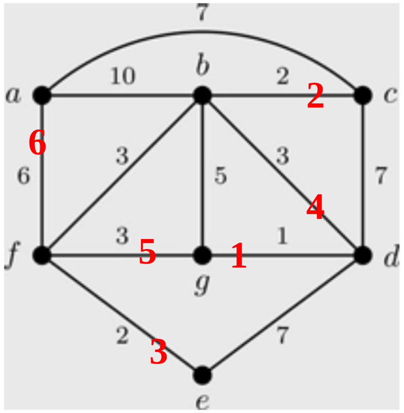


Minimum Spanning Trees

- **Definition 3.3** Given a weighted graph $G=(V,E,w)$, T is a minimum spanning tree, or MST, of G if it is a spanning tree with the least total weight.
- **Kruskal's Algorithm**
  - *Input*: Weighted connected graph $G=(V,E)$.
  - *Steps*:

    1. Choose the edge of least weight. Highlight it and add it to $T=(V,E')$.

    2. Repeat Step (1) so long as no circuit is created. That is, keep picking the edges of least weight but skip over any that would create a cycle in T.
  - *Output*: Minimum spanning tree T of G.
- **Example 3.2** Find the minimum spanning tree of the graph G below using Kruskal's Algorithm.



Minimum Spanning Trees

Prim's Algorithm

- *Input*: Weighted connected graph $G=(V,E)$.
- Steps:

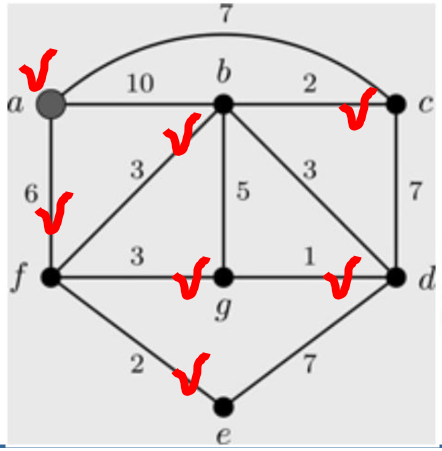
1. Let v be the root. If no root is specified, choose a vertex at random. Highlight it and add it to $T=(V',E')$.

2. Among all edges incident to v, choose the one of minimum weight. Highlight it. Add the edge and its other endpoint to T.

3. Let S be the set of all edges with exactly endpoint from $V(T)$. Choose the edge of minimum weight from S. Add it and its other endpoint to T.

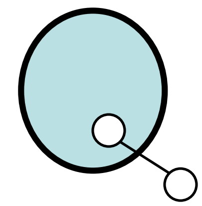
4. Repeat Step (3) until T contains all vertices of G, that is $V(T)=V(G)$.

- *Output*: Minimum spanning tree T of G.

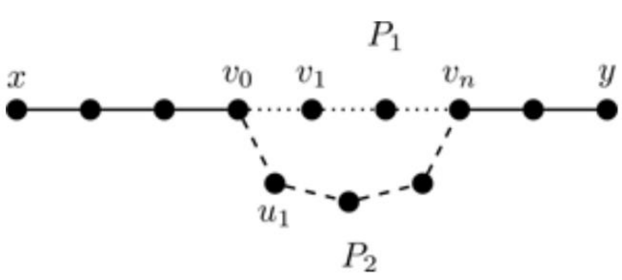


3.2 Tree Properties

- **Theorem 3.4** Every tree with at least two vertices has a leaf.
  - ✓ Prove by contradiction. Assume that there is no leaf.
- **Lemma 3.5** Given a tree T with a leaf v, the graph $T-v$ is still a tree.

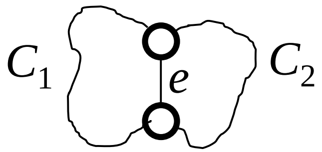


- **Theorem 3.6** A tree with n vertices has $n-1$ edges for all $n \geq 1$.
  - ✓ Prove by induction
- **Corollary 3.7** The total degree of a tree on n vertices is $2n-2$.
- **Proposition 3.8** Let T be a tree. Then for every pair of distinct vertices x and y there exists a unique $x-y$ path.
  - ✓ Prove by contradiction. Assume there are two paths P_1 and P_2 connecting x and y.
  - ✓ v_1 is the first vertex that is on P_1 but not in P_2. v_n is the next vertex on both P_1 and P_2.



Tree Properties

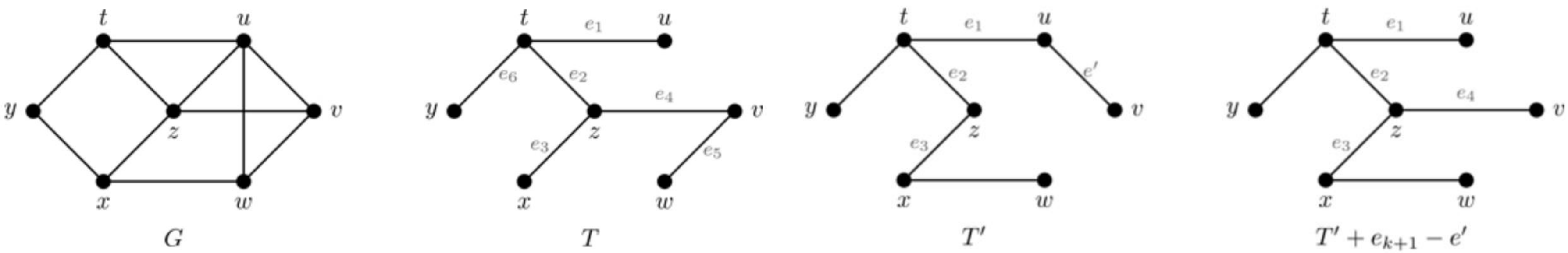
- **Proposition 3.9** Every tree is minimally connected, that is the removal of any edge disconnects the graph.
- **Proposition 3.10** If any edge e is added to a tree T then $T+e$ contains exactly one cycle.
  - ✓ $T+e$ contains at least one cycle
  - ✓ Assume two cycles C_1 and C_2 are in $T+e$



- **Theorem 3.11** Let T be a graph with n vertices. The following conditions are equivalent:
  - ✓ (a) T is a tree.
  - ✓ (b) T is acyclic and contains $n-1$ edges.
  - ✓ (c) T is connected and contains $n-1$ edges.
  - ✓ (d) There is a unique path between every pair of distinct vertices in T.
  - ✓ (e) Every edge of T is a bridge.
  - ✓ (f) T is acyclic and for any edge e from T, $T+e$ contains exactly one cycle.

Tree Properties

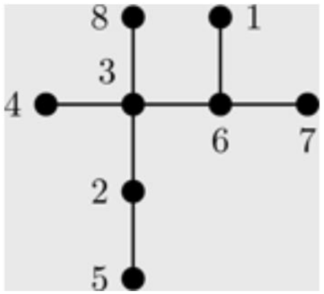
- **Theorem 3.12** Kruskal's Algorithm produces a minimum spanning tree.
  - ✓ Prove by contradiction. Let T' be the MST agreeing with the construction of T for the longest time ($e_1, \ldots, e_k$).
  - ✓ $w(e_{k+1}) \le w(e')$

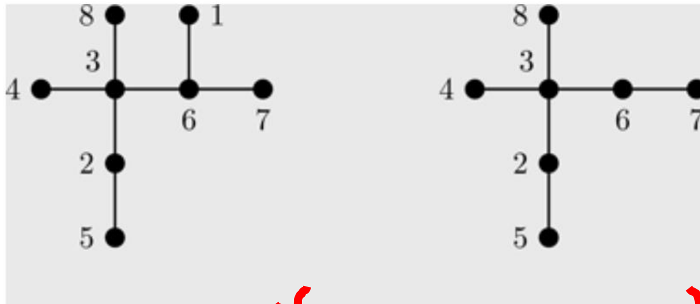


- **Tree Enumeration**
- **Definition 3.13** Given a tree T on n>2 vertices (labeled 1, 2,…, n), the *Prüfer sequence* of T is a sequence ($s_1, s_2, \ldots, s_{n\text{-}2}$) of length n−2 defined as follows:
  - ✓ Let l_1 be the leaf of T with the smallest label. Define T_1 to be $T-l_1$.For each $i \ge 1$, 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 .
  - ✓ The main idea is to prune the leaf of the smallest index, while keeping track of its unique neighbor.

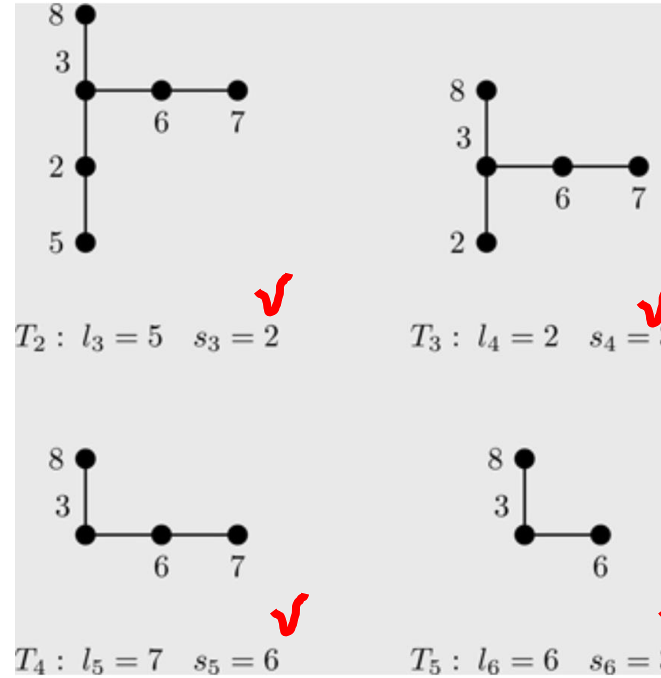
Tree Enumeration

- **Example 3.4** Find the Prüfer sequence for the tree below.





$T: l_1 = 1 \quad s_1 = 6$ √ $\quad T_1: l_2 = 4 \quad s_2 = 3$ √



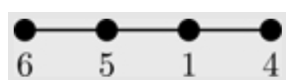
$T_2: l_3 = 5 \quad s_3 = 2$ √ $\quad T_3: l_4 = 2 \quad s_4 = 3$ √

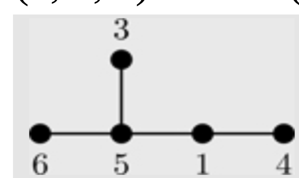
$T_4: l_5 = 7 \quad s_5 = 6$ √ $\quad T_5: l_6 = 6 \quad s_6 = 3$ √

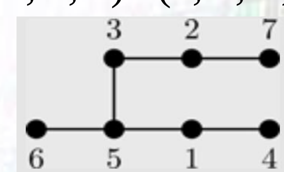
✓ Prüfer sequence: 6, 3, 2, 3, 6, 3

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

PS: (1, 5, 5, 3, 2)	(5, 5, 3, 2)	(5, 3, 2)	(3, 2)	(2)	
LF: (4, 6, 7)	(1, 6, 7)	(6, 7)	(5, 7)	(3, 7)	(2, 7)
DONE:	(4)	(1, 4)	(1, 4, 6)	(1, 4, 5, 6)	(1, 3, 4, 5, 6)





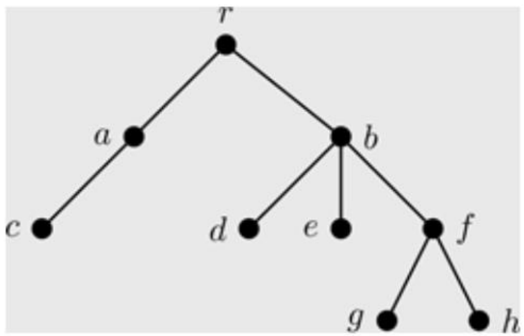


- **Theorem 3.14** (*Cayley's Theorem*) There are n^{n-2} different labeled trees on n vertices.

3.3 Rooted Trees

- **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.

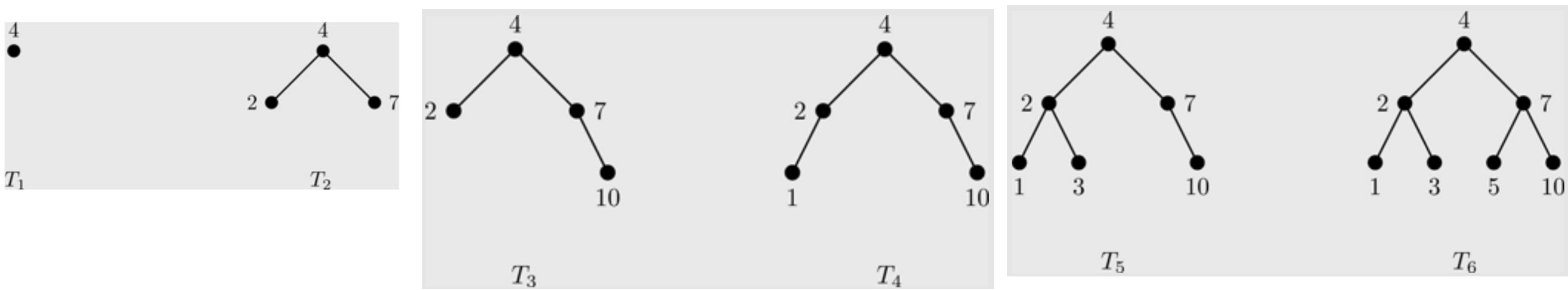
- **Example 3.6** Find the level of each vertex and the height of the rooted tree shown below.



- **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.

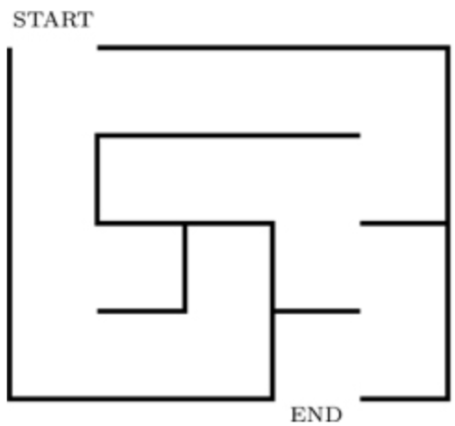
Rooted Trees

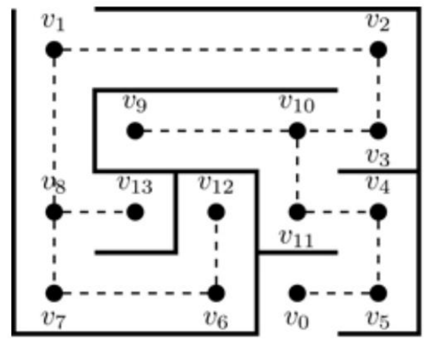
- **Definition 3.17** A tree in which every vertex has at most two children is called a *binary tree*. If every parent has exactly two children we have a *full binary tree*. Similarly, if every vertex has at most k children then the tree is called a *k-nary tree*.
- **Example 3.7** Trees can be used to store information for quick access. Consider the following string of numbers: 4, 2, 7, 10, 1, 3, 5



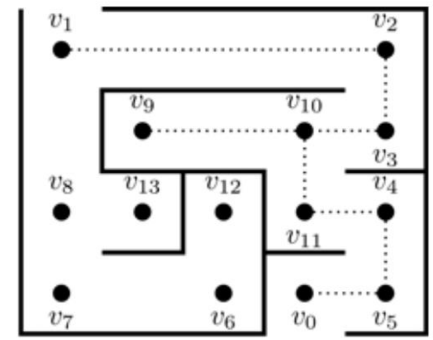
- **Theorem 3.18** Let T be a binary tree with height h and l leaves. Then (i) $l \leq 2h$. (ii) if T is a full binary tree and all leaves are at height h, then $l=2h$. (iii) if T is a full binary tree, then $n=2l-1$.

Maze

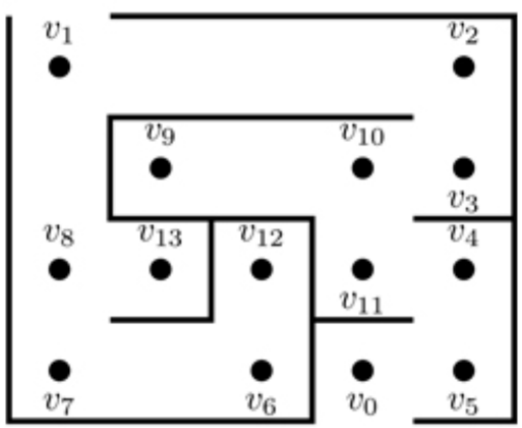


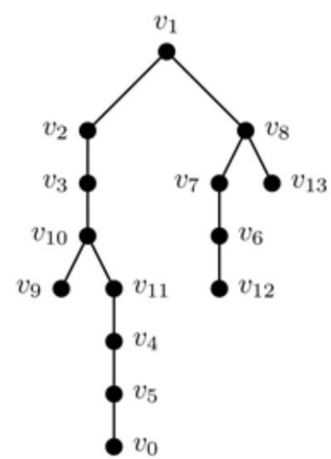


Breadth-First Search

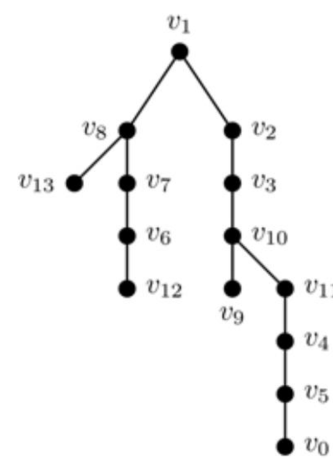


Depth-First Search





Maze Breadth-First Tree



Maze Depth-First Tree

3.4 Traveling Salesman Revisited

- A specific instance of the Traveling Salesman Problem when the weights assigned to the edges satisfy the triangle inequality;that is, for a weighted graph $G=(V, E, w)$, given any three vertices x, y, z we have $w(xy)+w(yz)\geq w(xz)$. This is also called *metric Traveling Salesman Problem* (*mTSP*)
- **mTSP Algorithm**
  - ✓ *Input*: Weighted complete graph K_n, where the weight function w satisfies the triangle inequality.
  - ✓ *Steps*:

    (1) Find a minimum spanning tree T for K_n.

    (2) Duplicate all the edges of T to obtain T^*.

    (3) Find an eulerian circuit for T^*.

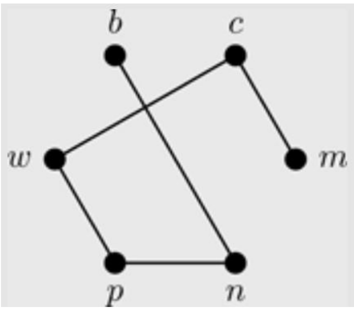
    (4) Convert the eulerian circuit into a hamiltonian cycle by skipping any previously visited vertex (except for the starting and ending vertex).

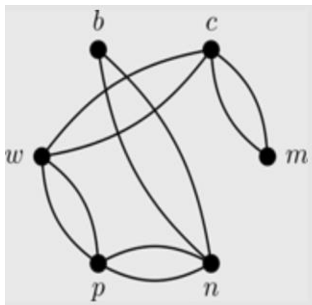
    (5) Calculate the total weight.
  - ✓ *Output*: hamiltonian cycle for K_n.

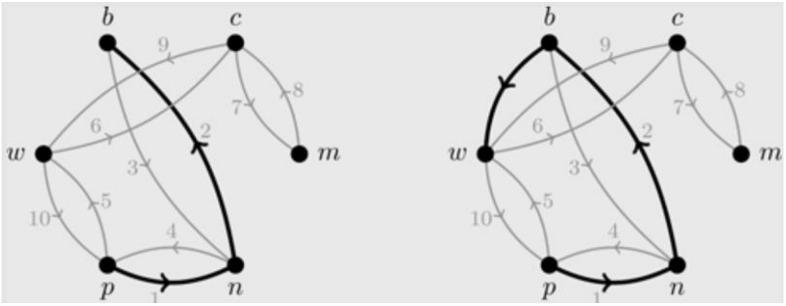
Traveling Salesman Revisited

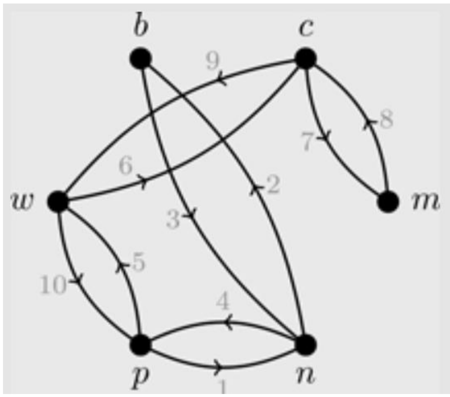
- **Example 3.10** Nour must visit clients in six cities next month and needs to minimize her driving mileage. The table below lists the driving distances between these cities. Use the mTSP Algorithm to find a good plan for her travels if she must start and end her trip in Philadelphia. Include the total distance.

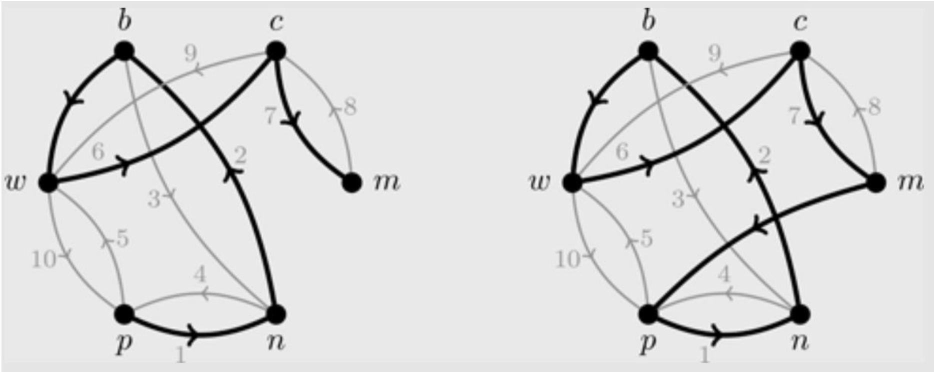
	Boston	Charlotte	Memphis	New York	Philadelphia	D.C.
Boston	·	840	1316	216	310	440
Charlotte	840	·	619	628	540	400
Memphis	1316	619	·	1096	1016	876
New York City	216	628	1096	·	97	228
Philadelphia	310	540	1016	97	·	140
Washington, D.C.	440	400	876	228	140	·











- ✓ 1472 for MST
- ✓ 2788 vs. 2781 (optimal) only a relative error of 0.25%

Supplement – Traveling Salesman Revisited

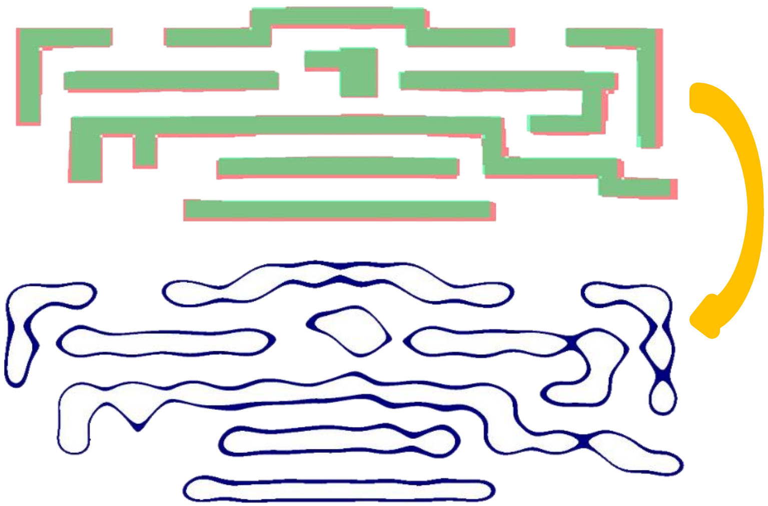
- **Theorem S.3.1.** *For all instances of the TSP that obey the triangle inequality, the solution produced by Algorithm mTSP is never worse than twice the optimal value.*
  - Let C be the hamiltonian cycle produced by Algorithm mTSP, and let C^* and T^* be a minimum-weight hamiltonian cycle and a minimum-weight spanning tree, respectively.
  - The total edge-weight of the eulerian tour is $2 \times wt(T^*)$, and since each shortcut is an edge that joins the initial and terminal points of a path of length at least 2, the triangle inequality implies that $wt(C) \leq 2 \times wt(T^*)$.
  - But C^* minus one of its edges is a spanning tree, which implies $2 \times wt(T^*) \leq 2 \times wt(C^*)$.

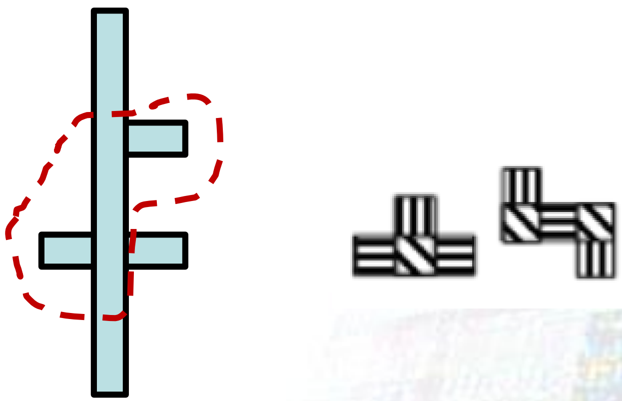
Supplement – Pruffer Encoding Application

- Serious image distortion in advanced technology nodes – Design for Manufacturability (DFM) issues
- Pattern calibration – identify hot-spot patterns according to a set of pattern library
- Problem definition: given a layout possibly consisting of more than hundreds of million polygons and a set of pattern library.
  - ✓ Identify all occurrences of each pattern in the layout without any false alarm.
  - ✓ A matching includes the case that a pattern matches partial set of a polygon.

Mask

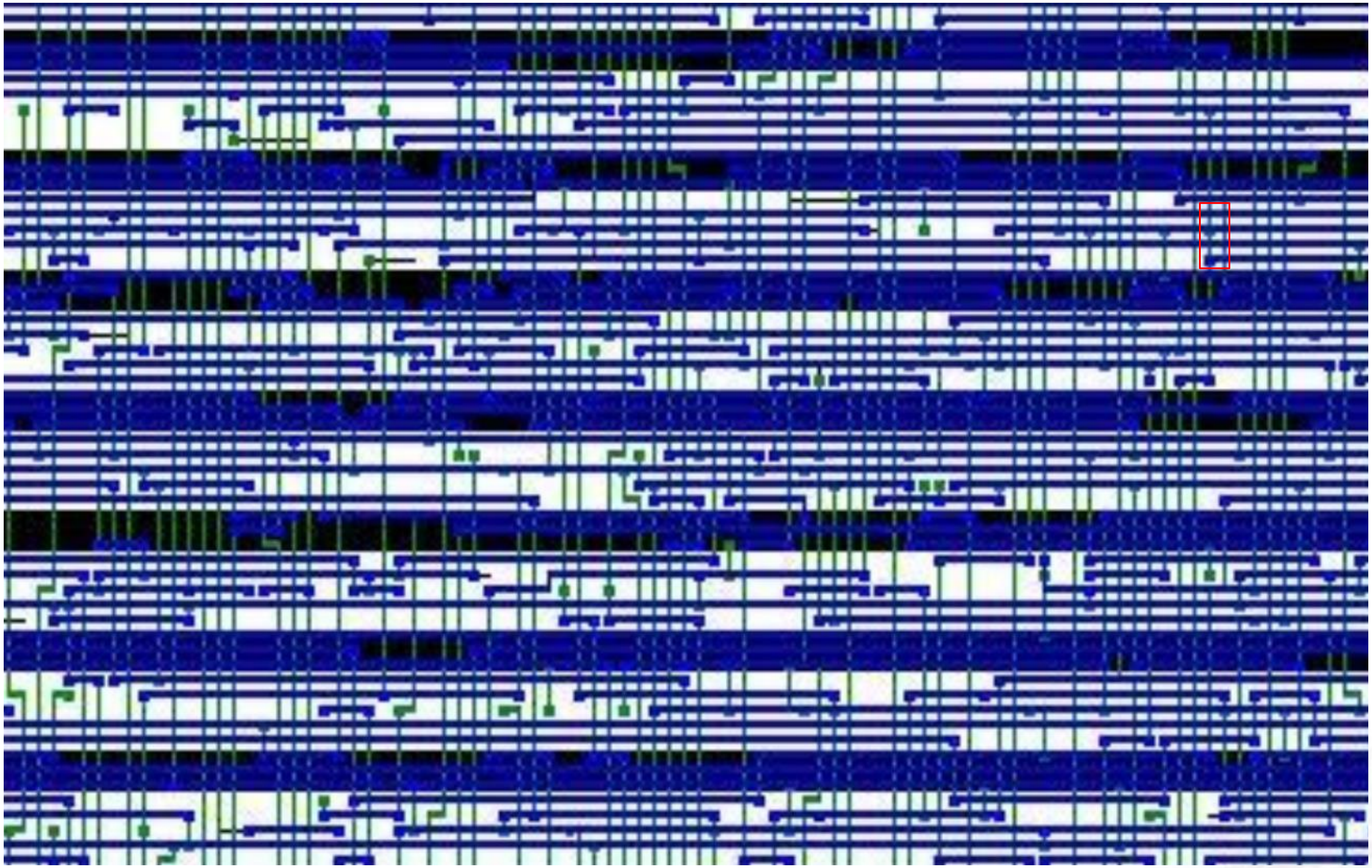
Result

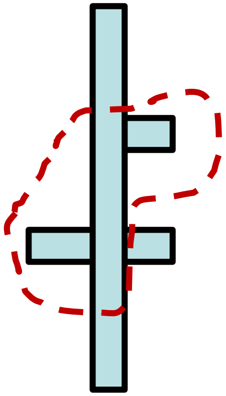


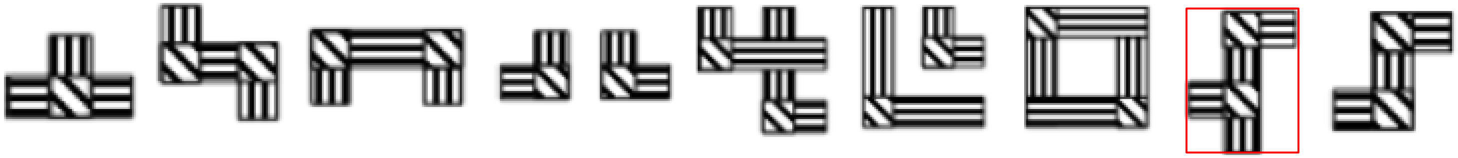


• Hong-Yan Su, Chieh-Chu Chen, Yih-Lang-Li, An-Chun Tu, Chuh-Jen Wu and Chen-Ming Huang, “A Novel Fast Layout Encoding Method for Exact Multi-Layer Pattern Matching with Prüfer-Encoding”, IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems, 2015.

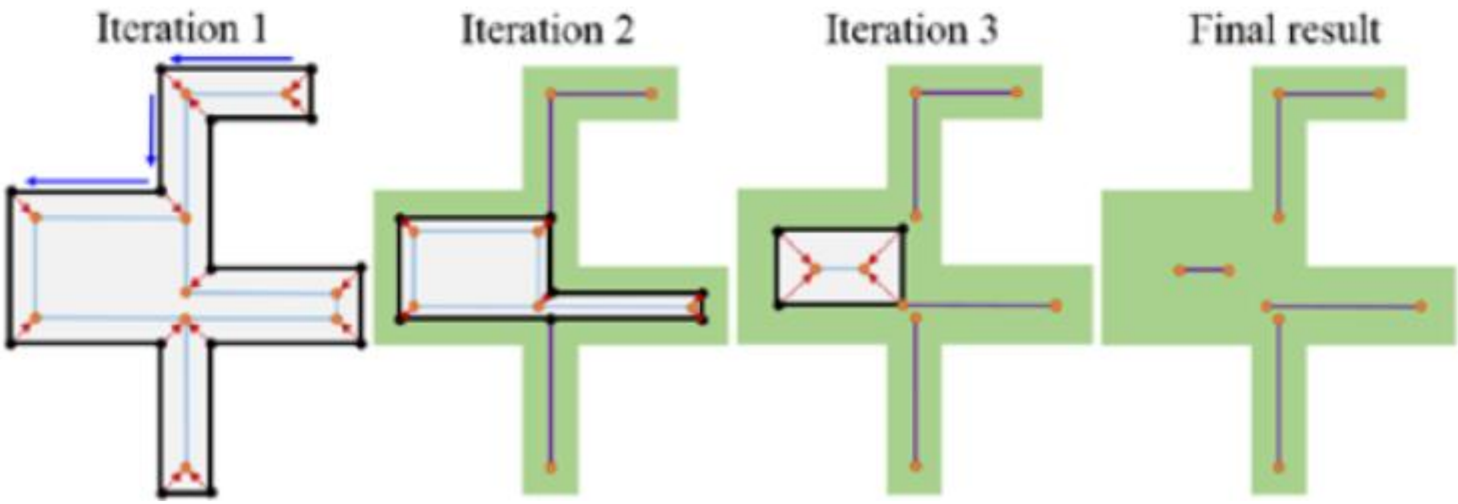
Supplement – Pruffer Encoding Application



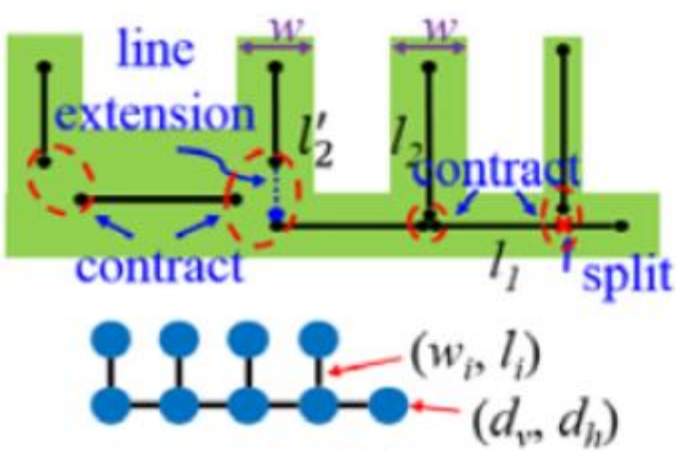




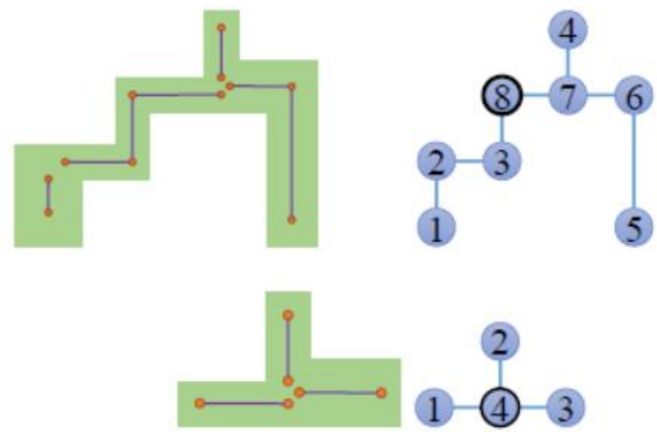
Supplement – Pruffer Encoding Application



Polygon to centerlines

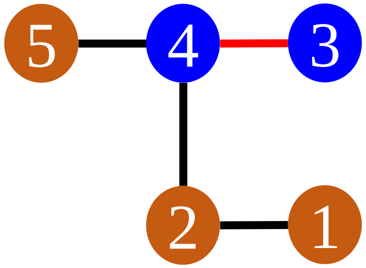


Centerlines to a tree

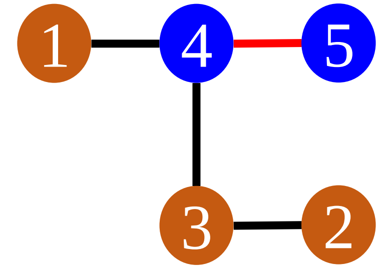


Partial recognition

Supplement – EPC Transformation Algorithm



f					
EPC_1	1	2	3	4	5
EPC_2	2	3	5	4	1



Q: How to find the function f ?

EPC_1				
L_d	1 → 2	2 → 3	3 → **5**	4 → **4**
C_{num}	2 → 3	4 → 4	4 → **4**	5 → **1**
D_{d2n}	*L*	*U*	*L*	*L*
DP_d	0,0	0,0	0,0	2,0

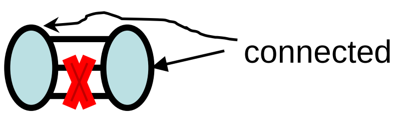
EPC_1				
L_d	2	3	4	1
C_{num}	3	4	5	4
D_{d2n}	*L*	*U*	***R***	***R***
DP_d	0,0	0,0	**2,0**	**0,0**

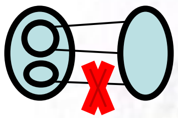
Supplement – Cycles, Edge-Cuts, Spanning Trees

- **Proposition S.3.2.** *A graph G is connected if and only if it has a spanning tree.*
  - *Necessity* ($\rightarrow$)
    - Assume T is the connected spanning subgraph of G with the least number of edges and T has cycles.
    - Then we can remove an edge in a cycle from T such that T is still a connected spanning subgraph. Contradiction to the edge minimality of T. Thus T has no cycle.
  - *Sufficiency* ($\leftarrow$) Spanning tree is connected, so G is connected.
- **Proposition S.3.3.** *A subgraph H of a connected graph G is a subgraph of some spanning tree if and only if H is acyclic.*
  - *Necessity* ($\rightarrow$) H is a spanning tree's subgraph, then H is acyclic by definition.
  - *Sufficiency* ($\leftarrow$)
    - Let H be an acyclic subgraph of G, and let T be any spanning tree of G.
    - Consider the connected, spanning subgraph G_1, where $V_{G1} = V_T \cup V_H$ and $E_{G1} = E_T \cup E_H$.
    - If G_1 is acyclic, then H is contained in T.
    - Otherwise suppose G_1 has a cycle C_1. Since H is acyclic, there is an edge e_1 in C_1 but not in H.
    - We can remove e_1 to get G_2 that is still a connected spanning subgraph of G and still contains H. If G_2 is acyclic, then we get a spanning tree, or repeat the same process to get an acyclic spanning tree of G.

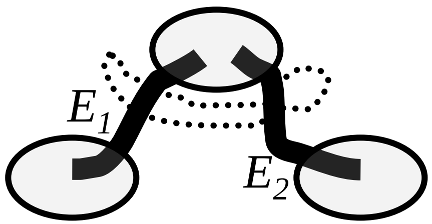
Supplement – Partition-Cuts and Minimal Edge-Cuts

- **DEFINITION:** An ***edge-cut*** in a graph G is a set of edges D such that G-D has more components than G.
- **DEFINITION**: Let G be a graph, and let X_1 and X_2 form a partition of V_G. The set of all edges of G having one endpoint in X_1 and the other endpoint in X_2 is called a ***partition-cut*** of G and is denoted $\langle X_1, X_2\rangle$.
- **Proposition S.3.4.** *Let $\langle X_1, X_2\rangle$ be a partition-cut of a connected graph G. If the subgraphs of G induced by the vertex sets X_1 and X_2 are connected , then $\langle X_1, X_2\rangle$ is minimal edge-cut.*
  - ✓ We have to prove (1) $\langle X_1, X_2\rangle$ is an edge cut and (2) any proper subset of $\langle X_1, X_2\rangle$ is not an edge cut, then $\langle X_1, X_2\rangle$ is a minimal edge cut.
  - ✓ (2) Assume S is any proper subset of $\langle X_1, X_2\rangle$, then there is at least one edge $e \in \langle X_1, X_2\rangle - S$ (e connects two connected subgraphs of $G - \langle X_1, X_2\rangle$).
  - ✓ Thus $G - S$ is connected, and any proper subset S of $\langle X_1, X_2\rangle$ is not a edge cut, implying $\langle X_1, X_2\rangle$ is a minimal edge cut.





Supplement – Partition-Cuts and Minimal Edge-Cuts

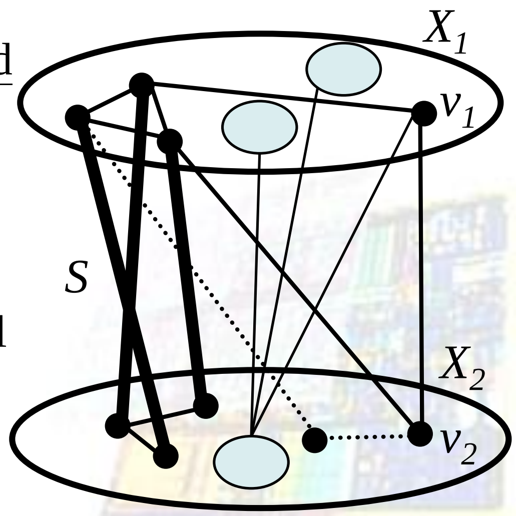


✓$E_1 \cup E_2$ is a partition cut as well as an edge cut, but not a minimal edge cut.
✓E_1 and E_2 are both minimal edge cuts and partition cuts.

- **Proposition S.3.5.** *A partition-cut $\langle X_1, X_2\rangle$ in a connected graph G is a minimal edge-cut of G or union of edge-disjoint minimal edge-cuts.*

  ✓

  Since $\langle X_1, X_2\rangle$ is an edge cut of G, it must contain a minimal edge cut, say S. If $\langle X_1, X_2\rangle \neq S$, let $e \in \langle X_1, X_2\rangle - S$, where the endpoints v_1 and v_2 of e lie in X_1 and X_2, respectively. Since S is a minimal edge cut, the X_1-endpoints of S are in one component of $G - S$, say S_1, and the X_2-endpoints of S are in the other component, say S_2. Furthermore, v_i and v_2 are in the same component of $G - S$ ($e \notin S$). Suppose without loss of generality, v_1 and v_2 are in S_1, then every path in G from v_1 to v_2 must use at least one edge of $\langle X_1, X_2\rangle - S$. Thus $\langle X_1, X_2\rangle - S$ is an edge cut of G and, hence, contains a minimal edge cut R. Repeat to check if $\langle X_1, X_2\rangle - (S \cup R)$ is empty. Eventually, we get $\langle X_1, X_2\rangle - (S_1 \cup S_2 \cup \ldots \cup S_r) = \phi$.

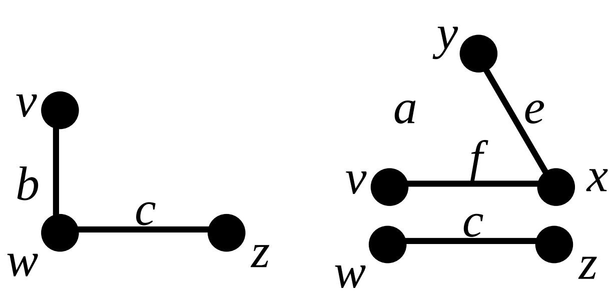


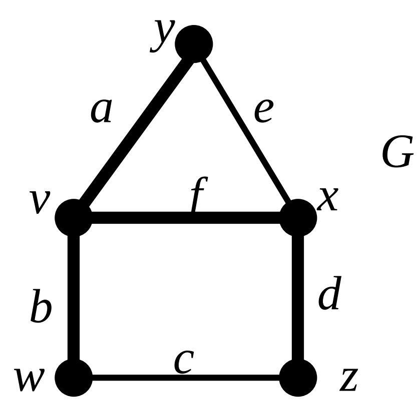
Supplement – Fundamental Cycles and Fundamental Edge-Cuts

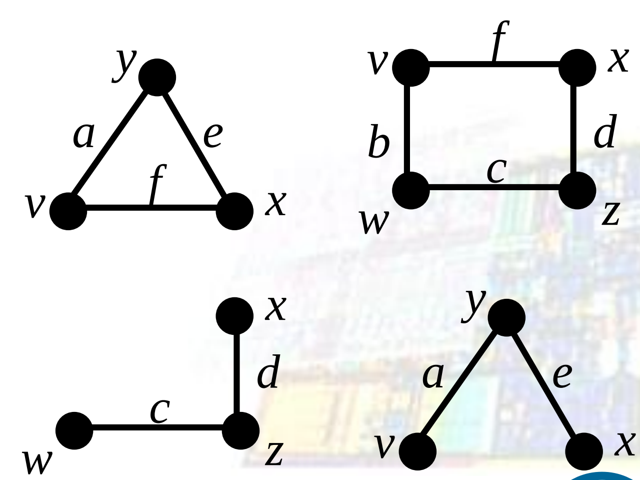
- DEFINITION: Let G be a graph with $c(G)$ components. The ***edge-cut rank*** of G is the number of edges in a full spanning forest of G. Thus, the edge-cut rank equals $|V_G| - c(G)$. (You can regard this as the number of edge cuts)
- DEFINITION: Let G be a graph with $c(G)$ components. The ***cycle rank*** (or ***Betti number***) of G, denoted $\beta(G)$, is the number of edges in the relative complement of a full spanning forest of G. Thus, the cycle rank is $\beta(G) = |E_G| - |V_G| + c(G)$. (You can regard this as the number of cycles or edge redundancies)
- **Remark**: Observe that *all* of the edges in the relative complement of a spanning forest could be removed without increasing the number of components. Thus, the cycle rank $\beta(G)$ equals the maximum number of edges that can be removed from G without increasing the number of components. Therefore, $\beta(G)$ is a measure of the *edge redundancy* with respect to the graph's connectedness.
- DEFINITION: Let F be a full spanning forest of a graph G, and let e be any edge in the relative complement of forest F. The cycle in the subgraph $F + e$ is called a ***fundamental cycle of*** G ***(associated with the spanning forest*** F***)***.
  - ✓ Fundamental cycle: one non-tree edge and the other are tree edges.

Supplement – Fundamental Cycles and Fundamental Edge-Cuts

- **Remark**: Each of the edges in the relative complement of a full spanning forest F gives rise to a *different* fundamental cycle.
- DEFINITION: The ***fundamental system of cycles*** associated with a full spanning forest F of a graph G is the set of all fundamental cycles of G associated with F.
- DEFINITION: Let F be a full spanning forest of a graph G, and let e be any edge of F. Let V_1 and V_2 be the vertex-sets of the two new components of the edge-deletion subgraph $F - e$. Then the partition-cut $\langle V_1, V_2 \rangle$, which is a minimal edge-cut of G by Proposition S.3.4, is called a ***fundamental edge-cut*** (***associated with*** F). (one tree edge + other non-tree edges)
- **Remark**: For each edge of F, its deletion gives rise to a different fundamental edge-cut.
- DEFINITION: The ***fundamental system of edge-cuts*** associated with a full spanning forest F is the set of all fundamental edge-cuts associated with F.
- **Example S.3.1.**







Supplement – Relationship Between Cycles and Edge-Cuts

- **Proposition S.3.6.** *Let S be a set of edges in a connected graph G. Then S is an edge-cut of G if and only if every spanning tree of G has at least one edge in common with S.*
  - ✓ By Proposition S.3.2, S is an edge cut if and only if $G - S$ contains no spanning tree of G, implying $\forall\ T_i$ of G, $S \cap T_i \neq \phi$.
- **Proposition S.3.7.** *Let C be a set of edges in a connected graph G. Then C contains a cycle if and only if the relative complement of every spanning tree of G has at least one edge in common with C.*
  - ✓ By Proposition S.3.3, edge set C contains a cycle if and only if C is not contained in any spanning tree of G, which means that the relative complement of every spanning tree of G has at least one edge in common with C.
- **Proposition S.3.8.** *A cycle and a minimal edge-cut of a connected graph have an even number of edges in common.*
  - ✓ A minimal edge cut partitions a vertex set into two subsets. If the vertices of a cycle are in the same subset, then they have no edge in common.
  - ✓ If the vertices of a cycle are in two subsets, as the cycle enters the other subset from one subset, it must return to the original subset. Thus the cycle must use even edges in edge cut to cross two subsets.

Supplement – Relationship Between Cycles and Edge-Cuts

- **Example S.3.1.** check the number of edges of three cycles in common with each minimal edge-cut in Fig. S.3.1.
- **Proposition S.3.9.** *Let T be a spanning tree of a connected graph, and let C be a fundamental cycle with respect to an edge e^* in the relative complement of T. Then the edge-set of cycle C consists of edge e^* and those edges of tree T whose fundamental edge-cuts contain e^*.*
  - ✓ Let $e_1, \ldots, e_k$ be the edges of T that, with e^*, make up the cycle C, and let S_i, be the fundamental edge-cut with respect to e_i, $1 \le i \le k$. We'll first prove that every S_i contains e^*. And then prove every S_j does not contain e^*, for $e_j \notin \{e_1, \ldots, e_k\}$.
  - ✓ Edge e_i is **the only edge of *T*** common to both C and S_i (by the definitions of C and S_i).
  - ✓ By Proposition S.3.8, C and S_i must contain an even number of edges, and, hence, there must be an edge in the relative complement of T that is also common to both C and S_i. But e^* is the only edge in the complement of T that is in C. Thus, the fundamental edge-cut S_i must contain e^*, $1 \le i \le k$.
    - ➢ e_i and e^* are two common edges to both C and S_i.
  - ✓ To complete the proof, we must show that no other fundamental edge-cuts associated with T contain e^*. So let S be the fundamental edge-cut with respect to some edge b of T, different from $e_1, \ldots, e_k$. Then S does not contain any of the edges $e_1, \ldots, e_k$ (by *S's* definition). The only other edge of cycle C is e^*, so by Proposition S.3.8, edge-cut S cannot contain e^*.

Supplement – Relationship Between Cycles and Edge-Cuts

- **Example S.3.1.** Verify Propositions S.3.8 and S.3.9 with cycle *a–e–f* any edge cut
- **Proposition S.3.10.** *The fundamental edge-cuts with respect to an edge e of a spanning tree T consists of e and exactly those edges in the relative complement of T whose fundamental cycles contain e.*
- **Theorem S.3.11 [Eulerian-Graph Characterization].** *The following statements are equivalent for a connected graph G.*

  1. G is eulerian. 2. The degree of every vertex in G is even.

  3. E_G is the union of the edge-sets of a set of edge-disjoint cycles of G.

  - ✓ (1→2) every pass-in to a vertex must follow a go-out to the vertex.
  - ✓ (2 →3) G is conneected and each vertex has even degree, so G must not be a tree. G has a cycle, say C_1. If $G_1 = C_1$, proof is complete. Otherwise let $G_1 = G – E_{C1}$. Since the degree of every vertex in C_1 is decreased by two, so every vertex in G_1 also has even degree. Thus G_1 also has a cycle…
  - ✓ (3→1)