Chap 3. Trees

Yih-Lang Li (李毅郎)

Computer Science Department

National Chiao Tung University, Taiwan

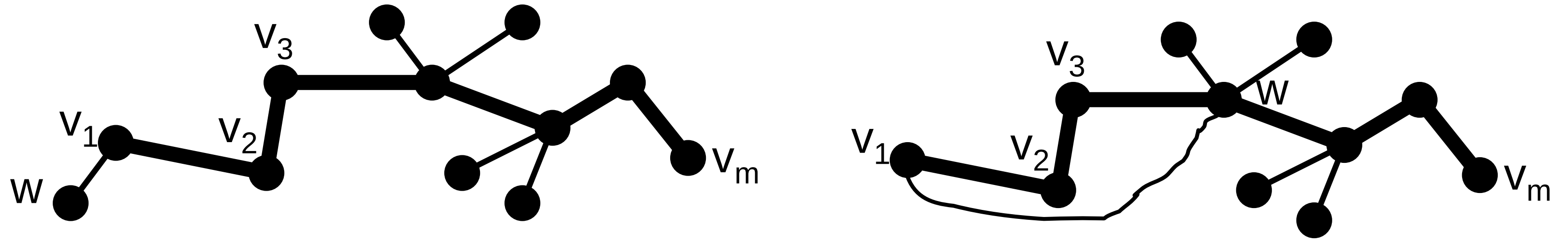
The sources of most figure images are from the course slides (Graph Theory) of Prof. Gross

Outline

- Characterizations and Properties of Trees
- Rooted Trees, Ordered Trees, and Binary Trees
- Binary-Tree Traversals (Skip)
- Binary-Search Trees (Skip)
- Huffman Trees and Optimal Prefix Codes
- Priority Trees
- Counting Labeled Trees: Prüfer Encoding
- Counting Binary Trees: Catalan Recursion

3.1 Characterizations and Properties of Trees

- **DEFINITION**: In an undirected tree, a ***leaf*** is a vertex of degree 1.
- **Proposition 3.1.1.** *Every tree with at least one edge has at least two leaves.*
  - ✓ Let $P=<v_1, \ldots, v_m>$ be a maximum-length path in a tree T. Consider one endpoint v_1. If $deg(v_1) = 1$, proved. If $deg(v_1) > 1$, case (a), v_1 connects to a vertex w not in $P \rightarrow P$ is not maximum-length path; case (b), v_1 connects to a vertex w in $P \rightarrow$ form a cycle.



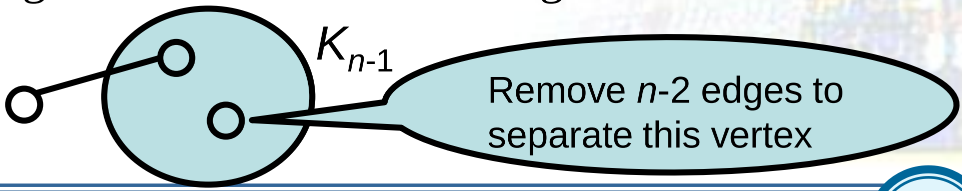
- **Corollary 3.1.2.** *If the degree of every vertex of a graph is at least 2, then that graph must contain a cycle.*
- **Proposition 3.1.3.** *Every tree on n vertices contains exactly $n - 1$ edges.*
  - ✓ Proof by induction. This holds for $k = 1$. Assume $k = n$, this also holds. For a tree $k = n+1$, we remove a leaf node v from T. $T - v$ is also a tree and only has n vertices, so $T - v$ contains exactly $n - 1$ edges. Thus T contains n edges.

Basic Properties of Trees

- **Corollary 3.1.4.** *A forest G on n vertices has n – c(G) edges.*
  - ✓ G has $c(G)$ trees, total edge = $\sum_{1\le i\le c(G)} |V_i| - 1 = n - c(G)$
- **Corollary 3.1.5.** *Any graph G on n vertices has at least n – c(G) edges.*
  - ✓ Each component contains at least ($|V_i|$ - 1) edges. Thus total at least n-$c(G)$ edges
- **Proposition 3.1.6.** *If G is a simple graph with n vertices and k components, then*

$$\left|E_G\right| \le \frac{(n-k)(n-k+1)}{2}$$

  - ✓ A simple graph with n vertices at most has $n(n-1)/2$ edges → edge upper bound is in proportional to n^2. *All* edges are distributed to k components. $100^2 > 99^2 + 1 > 98^2 + 2^2 > 90^2 + 10^2 > 90^2 + 5^2 + 5^2$. Thus edge upper bound is $(n-(k-1))(n-(k-1)-1)/2 = (n-k+1)(n-k)/2$
- **Corollary 3.1.7.** *A simple n-vertex graph with more than (n – 1)(n – 2)/2 edges must be connected.*
  - ✓ K_{n-1} contains ((n-1)(n-2)/2) edges. A simple graph with n vertices and more than $(n – 1)(n – 2)/2$ edges is constructed by adding one vertex and connecting this new vertex to any one vertex of K_{n-1}.



Six Different Characterizations of a Tree

- **Theorem 3.1.8.** *Let T be a graph with n vertices. Then the following statements are equivalent.*
  1. *T is a tree.*
  2. *T contains no cycles and has $n - 1$ edges.*
  3. *T is connected and has $n - 1$ edges.*
  4. *T is connected, and every edge is a cut-edge.*
  5. *Any two vertices of T are connected by exactly one path.*
  6. *T contains no cycles, and for any new edge e, the graph $T + e$ has exactly one cycle.*

  ✓ $1 \rightarrow 2$, by Proposition 3.1.3.

  ✓ $2 \rightarrow 3$, Assume T has k components, by Corollary 3.1.4, T has $n - k$ edges. $\rightarrow k = 1$.

  ✓ $3 \rightarrow 4$, T-e has $(n$-$2)$ edges. By Corollary 3.1.5, $(n-2) \geq n-c(T-e) \rightarrow c(T-e) \geq 2$.

  ✓ $4 \rightarrow 5$, every edge can not lie in a cycle, so any two vertices is connected by one path.

  ✓ $5 \rightarrow 6$, if T+e (e connects u and v) has two cycles, then T must has 2 paths from u to v.

  ✓ $6 \rightarrow 1$, we have to prove T has no cycle and is connected. Assume u and v are not connected, then T+uv does not have a cycle, contradiction to 6.

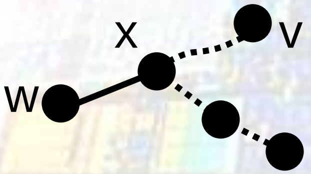
The Center of a Tree

- For a graph, the center $Z(G)$ can be anything, from a vertex to G. However, C. Jordan showed in 1869 that the center of a tree has only two possible cases.
- **Lemma 3.1.9.** *Let T be a tree with at least three vertices.*

  (*a*) *If v is a leaf of T and w is its neighbor, then* $ecc(v) = ecc(w) + 1$.

  (*b*) *If v is a central vertex of T, then* $deg(v) \geq 2$.

  - ✓ (a). A path from leaf v to any vertex must also pass its neighbor w.
  - ✓ (b). By (a), any leaf's *ecc* can not be minimum, so a center vertex's degree ≥ 2.
- **Lemma 3.1.10.** *Let v and w be two vertices in a tree T such that w is of maximum distance from v* (*i.e.*, $ecc(v) = d(v,w)$). *Then w is a leaf.*
  - ✓ if w is not a leaf, then w has a neighbor u not in the path from v to $w \rightarrow d(v,u) > d(v,w)$
- **Lemma 3.1.11.** *Let T be a tree with at least three vertices, and let T* be the subtree of T obtained by deleting from T all its leaves. If v is a vertex of T*, then* $ecc_T(v) = ecc_{T^*}(v) + 1$.
  - ✓ By Lemma 3.1.10, the endpoints of all longest paths from v are leaves. Let w be a leaf in T and $d(v,w) = ecc_T(v)$ and x be the neighbor of w. If $deg(x) > 2$, then the neighbor of x not in the path v–w must also be a leaf. In T^*, x becomes a leaf and $d(v,x) = ecc_{T^*}(v) = ecc_T(v) - 1$.

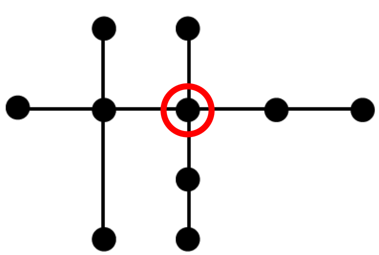


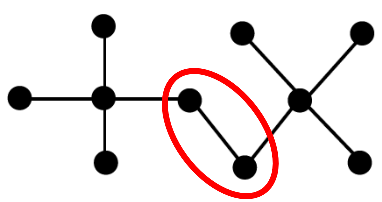
The Center of a Tree

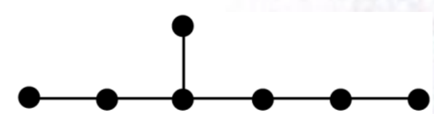
- **Proposition 3.1.12.** *Let T be a tree with at least three vertices, and let T* be the subtree of T obtained by deleting from T all its leaves. Then* $Z(T) = Z(T^*)$
  - ✓ By previous Lemma, the eccentricity of every vertex in T^* is less than that in T by one.
- **Corollary 3.1.13 [Jordan, 1869].** *Let T be an n-vertex tree. Then the center* $Z(G)$ *is either a single vertex or a single edge.*
  - ✓ We can iteratively delete the leaves of a tree until the new tree is a vertex or an edge. By Proposition 3.1.12. Original tree's center is a vertex or an edge.

Tree Isomorphisms and Automorphisms

- A center of a graph must be mapped to the other graph's center and the leaf and its image leaf must be the same distance from their respective centers.
- **Example 3.1.1 & 3.1.2:**







A tree that has no non-trivial automorphisms.

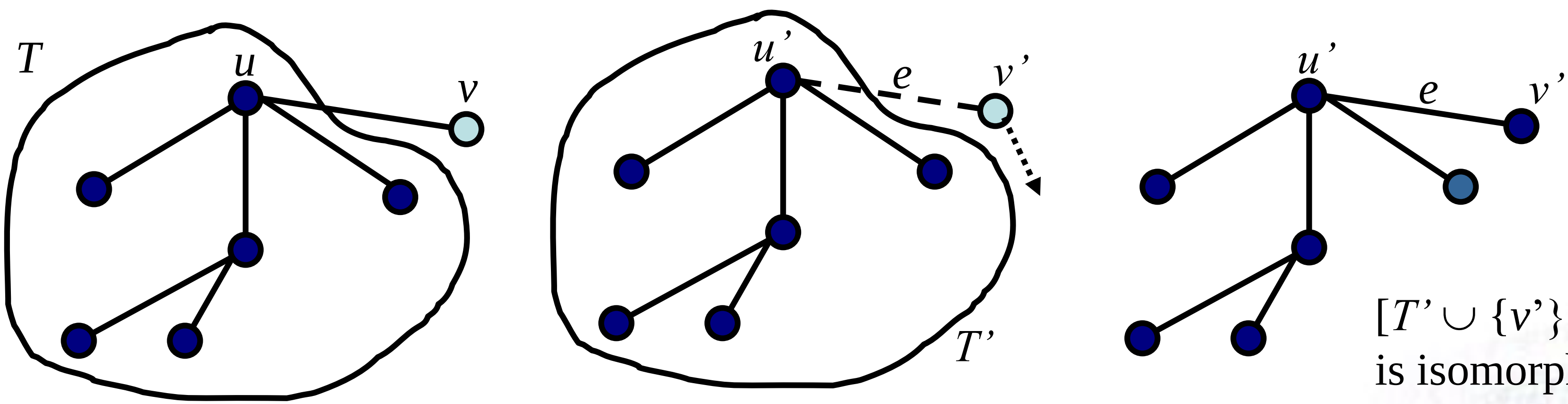
Tree-Graphic Sequences

- **DEFINITION**: A sequence $<d_1, d_2, \ldots, d_n>$ is said to be ***tree-graphic*** if there is a permutation of it that is the degree sequence of some n-vertex tree.
- **Theorem 3.1.14.** *A sequence* $<d_1, d_2, \ldots, d_n>$ *of* $n \geq 2$ *positive integers is tree-graphic if and only if* $\sum_{1 \le i \le n} d_i = 2n - 2$
  - ✓ $\rightarrow$, By Euler degree sum theorem & Proposition 3.1.3, degree sum = $2|E_T| = 2n - 2$
  - ✓ $\leftarrow$, By induction. This is true for n=2. It holds for n=k. As for n=k+1, $< d_1, d_2, \ldots, d_{k+1}>$ satisfies the condition that degree sum = $2(k+1) - 2 = 2k$. Assume $d_1 \geq d_2 \geq \ldots \geq d_{k+1}$. By simple counting arguments that $2 \leq d_1 \leq k$ and $d_k = d_{k+1} = 1$. The sequence $<d_1-1, d_2, \ldots, d_k>$ is positive and sum to $2k$-2, hence, there is a k-vertex tree T whose degree sequence is a permutation of $<d_1-1, d_2, \ldots, d_k>$. Let T^* be the tree by adding a new vertex to a vertex of T of degree d_1-1. Then the degree sequence of T^* is a permutation of the sequence $< d_1, d_2, \ldots, d_{k+1} >$.
- **NOTATION**: The minimum degree of the vertices of a graph G is denoted $\delta_{min}(G)$.
- **Theorem 3.1.15.** *Let T be any tree on n vertices, and let G be a simple graph such that* $\delta_{min}(G) \geq n - 1$. *Then T is a subgraph of G.*

Trees as Subgraphs

- **Theorem 3.1.15.** *Let T be any tree on n vertices, and let G be a simple graph such that* $\delta_{min}(G) \geq n-1$. *Then T is a subgraph of G.*
  - ✓ it holds as n=1 or 2 since K_1 and K_2 are subgraphs of every graph having at least one edge.
  - ✓ Assume it holds for some $n \geq 2$. Let T be a tree on n+1 vertices, and let G be a graph with $\delta_{min}(G) \geq n$. We have to show T is a subgraph of G.

- T

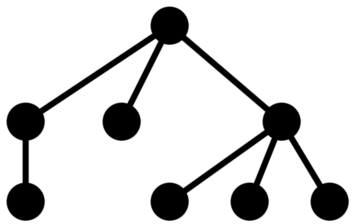


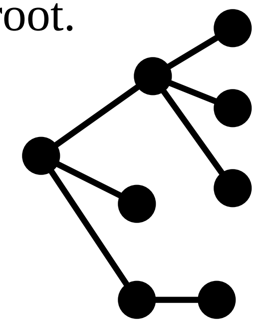
$[T' \cup \{v'\}] + e$ is isomorphic to T

T' in G is isomorphic to T-v.
$deg_G(u') \geq n$, there exists v' not in T' connecting u' with e

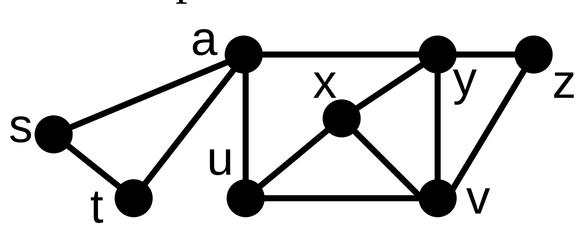
3.2 Rooted Trees, Ordered Trees, and Binary Trees

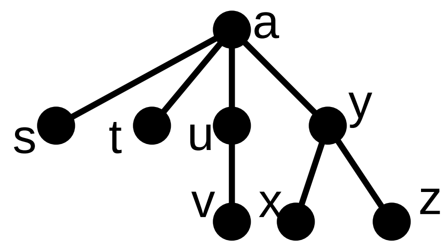
- **DEFINITION**: A ***directed tree*** is a directed graph whose underlying graph is a tree.
- **DEFINITION**: A ***rooted tree*** is a tree with a designated vertex called the root. Each edge is considered to be directed away from the root.





- **DEFINITION**: Let v be a vertex in a connected graph G. A ***shortest-path tree*** for G from v is a rooted tree T with vertex-set V_G and root v such that the unique path in T from v to each vertex w is a shortest path in G from v to w.

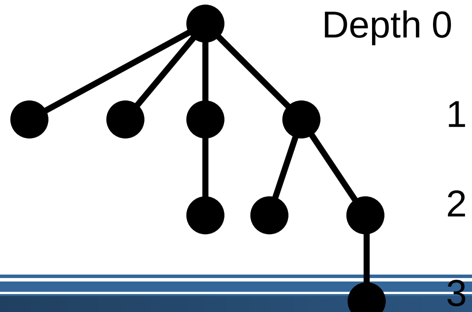




- **Remark:** The *breadth-first search* produces a shortest path tree for an unweighted graph, and *Dijkstra's algorithm* produces one for a weighted graph.

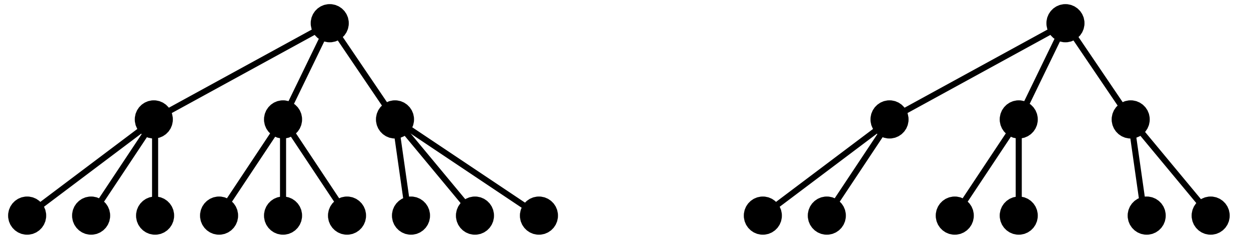
Rooted Tree Terminology

- **DEFINITION**: In a rooted tree, the ***depth*** or ***level*** of a vertex v is its distance from the root, i.e., the length of the unique path from the root to v. Thus, the root has depth 0.
- **DEFINITION**: The ***height*** of a rooted tree is the length of a longest path from the root (or the greatest depth in the tree).
- **DEFINITION**: If vertex v immediately precedes vertex w on the path from the root to w, then v is the ***parent*** of w and w is the ***child*** of v.
- **DEFINITION**: Vertices having the same parent are called ***siblings***.
- **DEFINITION**: A vertex w is called a ***descendant*** of a vertex v (and v is called an ***ancestor*** of w), if v is on the unique path from the root to w. If, in addition, $w \neq v$, then w is a ***proper*** descendant of v (and v is a proper ancestor of w).
- **DEFINITION**: A ***leaf*** in a rooted tree is any vertex having no children.
- **DEFINITION**: An ***internal vertex*** in a rooted tree is any vertex that has at least one child. The root is internal, unless the tree is trivial (i.e., a single vertex).



Rooted Tree Terminology

- **DEFINITION**: An ***m-ary tree*** ($m \geq 2$) is a rooted tree in which every vertex has *m* or fewer children.
- **DEFINITION**: A ***complete m-ary tree*** is an *m*-ary tree in which every internal vertex has exactly *m* children and all leaves have the same depth.



Two 3-ary trees, one complete and the other not complete

- **Proposition 3.2.1.** *A complete m-ary tree has m^k vertices at level k.*
  - ✓ The statement is trivially true for $k = 1$.

    Assume as an induction hypothesis that there are m^l vertices at level $k=l$, for some $l \geqq 1$.

    Since each of these vertices has *m* children, there are $m \cdot m^l = m^{l+1}$ children at level $l + 1$.
- **Corollary 3.2.2.** *An m-ary tree has at most m^k vertices at level k.*

Rooted Tree Terminology

- **Theorem 3.2.3.** *Let T be an n-vertex m-ary tree of height h. Then*

$$h+1 \le n \le \frac{m^{h+1}-1}{m-1}$$

  - ✓ Let n_k be the number of vertices at level k, so that $1 \le n_k \le m^k$, by Corollary 3.2.2. Thus,

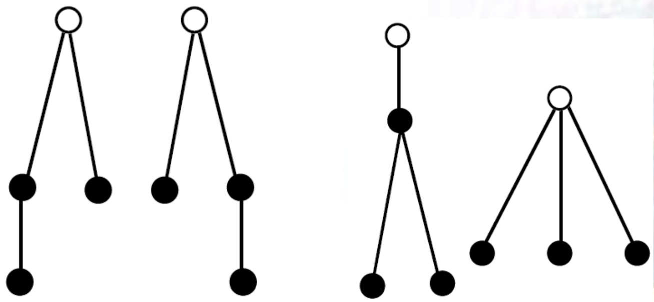
$$h+1 = \sum_{k=0}^{h} 1 \le \sum_{k=0}^{h} n_k \le \sum_{k=0}^{h} m^k = \frac{m^{h+1}-1}{m-1}$$

  The result follows since $\sum_{k=0}^{h} n_k = n$

- **Corollary 3.2.4.** *The complete m-ary tree of height h has* $\frac{m^{h+1}-1}{m-1}$ *vertices.*

Isomorphism of Rooted Trees

- **DEFINITION**: Two rooted trees are said to be ***isomorphic as rooted trees*** if there is a graph isomorphism between them that maps root to root.
- **DEFINITION**: An ***ordered tree*** is a rooted tree in which the children of each vertex are assigned a fixed ordering.



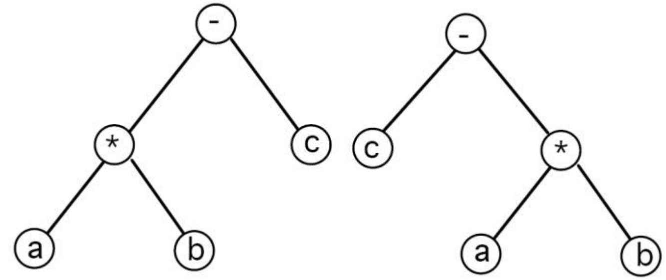
Ordered Trees

- **DEFINITION**: In a ***standard plane drawing*** of an ordered tree,

  (1) the root is at the top,

  (2) the vertices at each level are horizontally aligned,

  (3) the left-to-right order of the vertices agrees with their prescribed order.

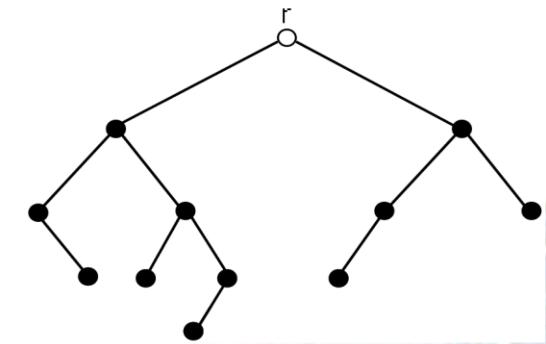


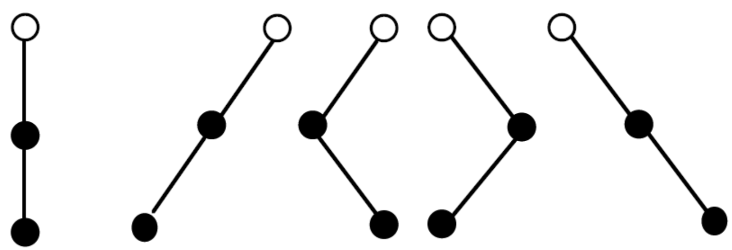
a * b - c   c - a * b

Binary Trees

- **DEFINITION**: A ***binary tree*** is an ordered 2-ary tree in which each child is designated either a ***left-child*** or a ***right-child***.

- **DEFINITION**: The ***left (right) subtree*** of a vertex *v* in a binary tree is the binary subtree spanning the left (right)-child of *v* and all of its descendants.



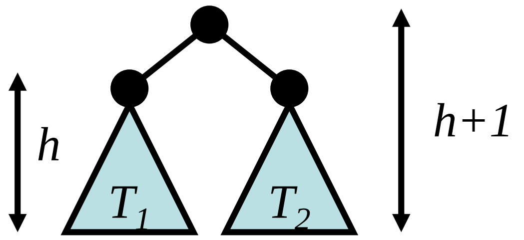


Four different binary trees that are the same ordered tree

Binary Trees

- RECURSIVE PROPERTY OF A BINARY TREE: If T is a binary tree of height h, then its left and right subtrees both have heights less than or equals to h-1, and equality holds for at least one of them.
- **Theorem 3.2.5.** *The complete binary tree of height h has* $2^{h+1} - 1$ *vertices.*
  - ✓ Both T_1 and T_2 have $(2^{h+1} - 1)$ vertices

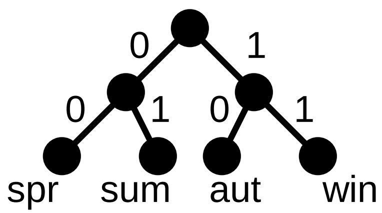
  Totally $2\times(2^{h+1} - 1) + 1 = 2^{h+2} - 1$



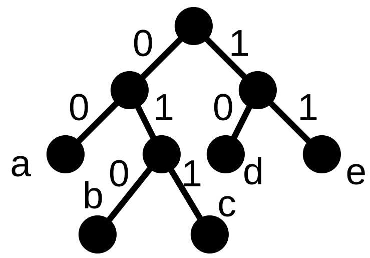
- **Corollary 3.2.6.** *Every binary tree of height h has at most* $2^{h+1} - 1$ *vertices.*

3.5 Huffman Trees and Optimal Prefix Codes

- **DEFINITION**: A ***binary code*** is an assignment of symbols or other meanings to a set of bitstrings. Each bitstring is referred to as a ***codeword***.
- **DEFINITION**: A ***prefix code*** is a binary code with the property that no codeword is an initial substring of any other codeword.
- **Application 3.5.1** *Constructing Prefix Codes by Binary Tree*:



- **Example 3.5.1**



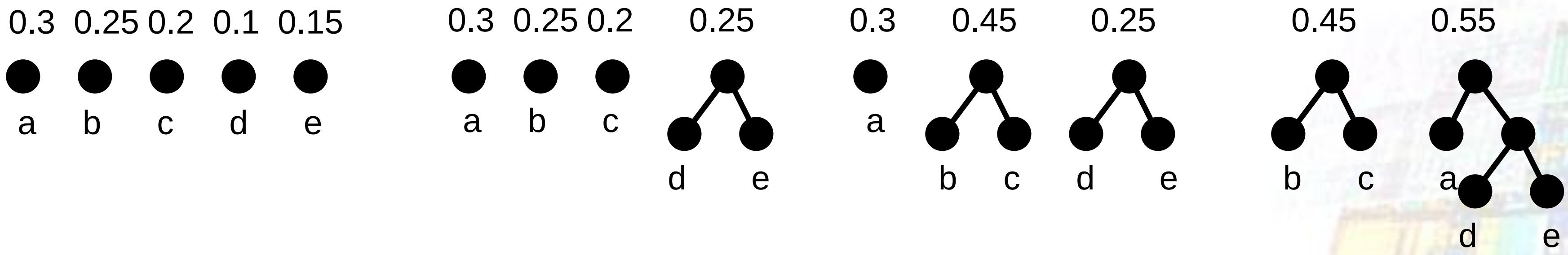
a: 00
b: 010
c: 011
d: 10
e: 11

Huffman Codes

- In a prefix code that uses its shorter codewords to encode the more frequently occurring symbols, the messages will tend to require fewer bits than in a code that does not .
  - ✓ This suggests that one measure of a code's efficiency be the *average weighted length* of its codewords.
- **Example 3.5.2.** Consider the codewords: a: 00 (0.3), b: 010 (0.25), c: 011 (0.2), d: 10 (0.1) , e: 11 (0.15). The average weighted length of a codeword is $2\times0.3+3\times0.25+3\times0.2+2\times0.1+2\times0.15=2.45$
- **DEFINITION**: Let T be a binary tree with leaves $s_1, s_2, \ldots, s_l$, such that each leaf s_i is assigned a weight w_i. Then the ***average weighted depth*** of the binary tree T, denoted $wt(T)$, is given by

$$wt(T) = \sum_{i=1}^{l} depth(s_i) \cdot w_i$$

- **Application 3.5.2.** *Constructing Efficient Codes – the Huffman Algorithm:*

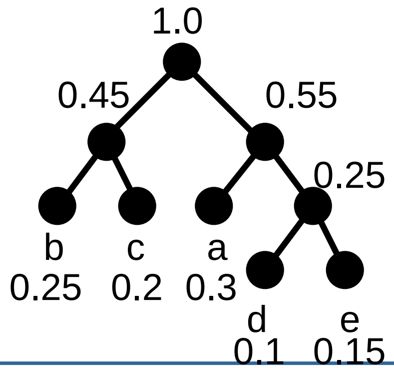


Huffman Codes

- **COMPUTATIONAL NOTE:** In choosing the two trees of smallest weights, ties are resolved by some default ordering of the trees in the forest.
- **DEFINITION**: The binary tree produced from Algorithm 3.5.1 is called the ***Huffman tree*** for the list of symbols, and its corresponding prefix code is called the ***Huffman code***.

- **Lemma 3.5.1**: *If the leaves of a binary tree are assigned weights, and if each internal vertex is assigned a weight equal to the sum of its children's weights, then the trees's average weighted depth equals the sum of the weights of its internal vertices.*

- **Theorem 3.5.2:** *For a given list of weights* $w_1, w_2, \ldots, w_l$*, a Huffman tree has the smallest possible average weighted depth among all binary trees whose leaves are assigned those weights.*

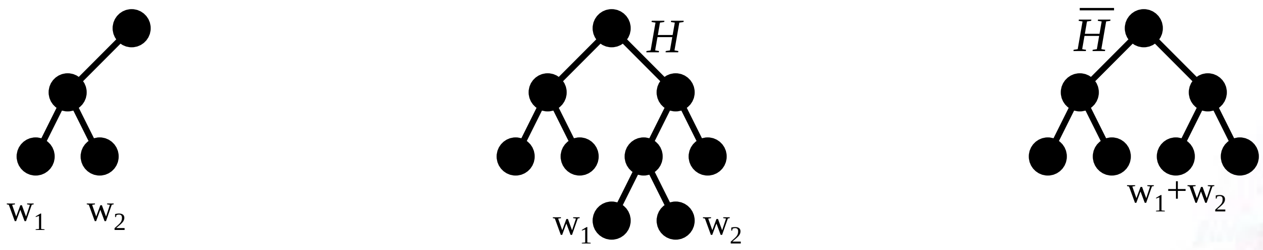


2*0.25 + 2*0.2 + 2*0.3 + 3*0.1 + 3*0.15 = <u>2.25</u>
1 + 0.45 + 0.55 + 0.25 = <u>2.25</u>

Huffman Codes

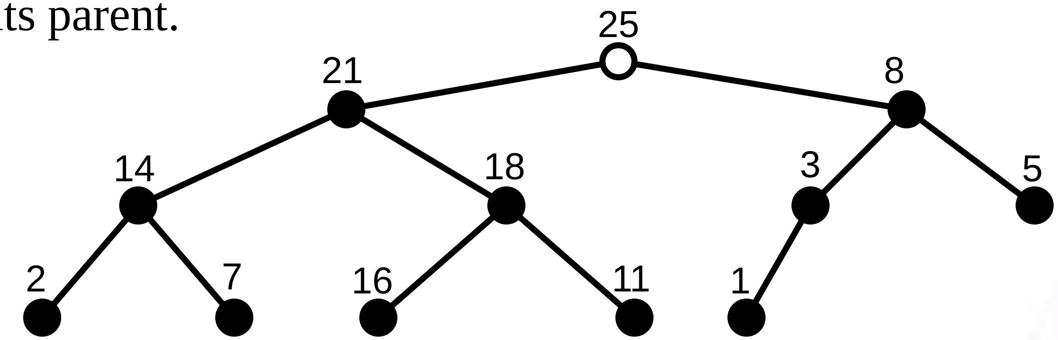
- By induction. Assume for some $l \geq 2$, Theorem holds.
- Let $w_1, w_2, \ldots, w_{l+1}$ be any list of $l+1$ weights, and assume w_1 and w_2 are two of the smallest ones. H is the Huffman tree, and $\overline{H}$ be the right one obtained from H.
  $wt(H) = wt(\overline{H}) + w_1 + w_2$. $\overline{H}$ is also a Huffman tree of l weights $\rightarrow$ $\overline{H}$ is optimal.
- Suppose T^* is an optimal binary tree for the weights $w_1, w_2, \ldots, w_{l+1}$. Let x be the internal vertex of T^* of greatest depth whose two descendants are leaves y and z of weights w_1 and w_2 (otherwise T* is not optimal).
- Let $\overline{T}$ be the tree by deleting y and z from T^*.

$$wt(T^*) = wt(\overline{T}) + w_1 + w_2. \quad wt(\overline{T}) \geq wt(\overline{H}). \quad wt(T^*) \geq wt(H).$$



3.6 Priority Trees

- **DEFINITION**: A binary tree of height h is called ***left-complete*** if the bottom level has no gaps as one traverses from left to right. More precisely, it must satisfy the following three conditions.
  - ✓ Every vertex of depth h - 2 or less has two children.
  - ✓ There is at most one vertex v at depth h – 1 that has only one child (a left one).
  - ✓ No vertex at depth h – 1 has fewer children than another vertex at depth h – 1 to its right.
- **DEFINITION**: A ***priority tree*** is a left-complete binary tree whose vertices have labels (*called priorities*) from an ordered set (or sometimes, a *partially ordered set*), such that no vertex has higher priority than its parent.



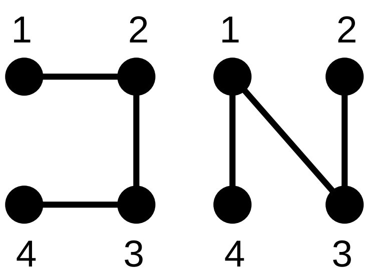
- **DEFINITION**: A ***priority queue*** is a set of entries, each of which is assigned a *priority*. When an entry is to be removed, or *dequeued*, from the queue, an entry with the highest priority is selected.

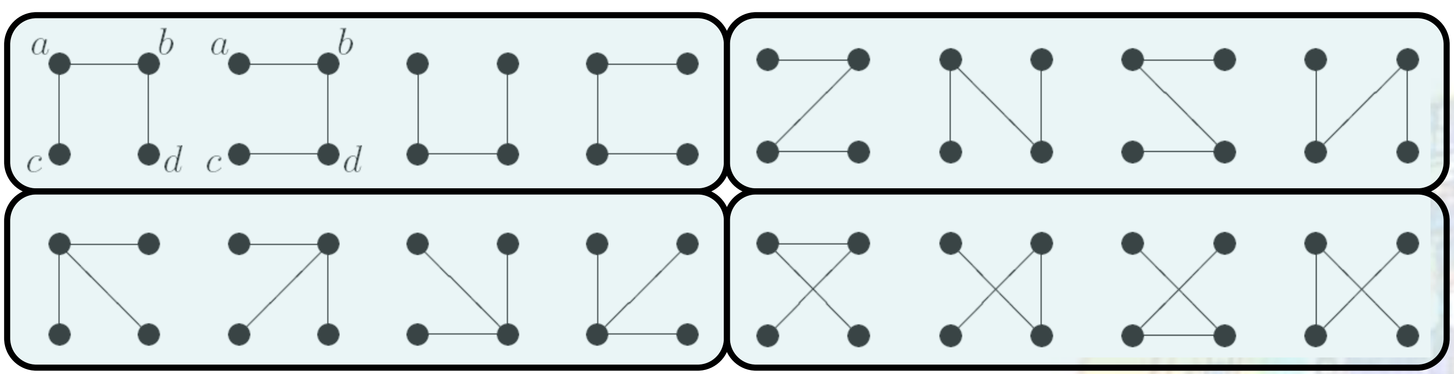
Heaps

- **DEFINITION**: A ***heap*** is a representation of a priority tree as an array, having the following address pattern.
  - ✓ $index(root) = 0$
  - ✓ $index(leftchind(v)) = 2 \times index(v) + 1$
  - ✓ $index(rightchind(v)) = 2 \times index(v) + 2$
  - ✓ $index(parent(v)) = \left\lfloor \dfrac{index(v) - 1}{2} \right\rfloor$

3.7 Counting Labeled Trees: Prüfer Encoding

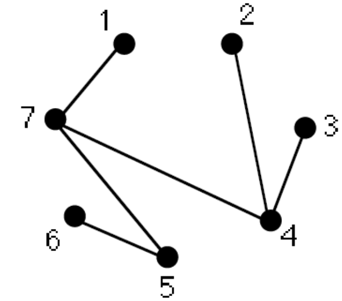
- **In 1875, Cayley presented a paper describing a method for counting certain hydrocarbons containing a given number of carbon atoms.**
  - ✓ **Also count the number of *n*-vertex trees with the standard vertex labels 1, …, *n*.**
- Two labeled trees are considered the same if their respective edge-sets are identical.
- **The number of *n*-vertex labeled trees is n^{n-2}, for $n \geq 2$, and is known as Cayley's Formula.**
- **Remark:** Counting the number of isomorphically distinct labeled *n*-vertex simple graphs is much more difficult. The Pólya-Burnside enumeration method, which is presented in Chapter 14, can be used to solve this kind of problem.





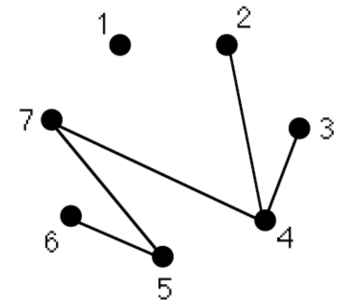
Prüfer Encoding

- **DEFINITION:** A **Prüfer sequence** of length $n - 2$, for $n \geq 2$, is any sequence of integers between 1 and n, with repetitions allowed.



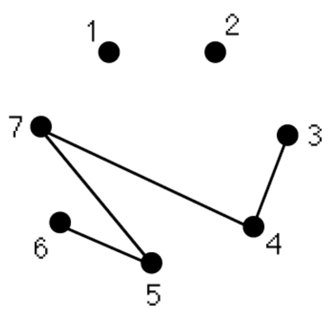
ALGORITHM: PRÜFER ENCODING
Input: an n-vertex tree with std 1-based vertex-labels.
Output: a Prüfer sequence of length $n - 2$.
Initialize T to be the given tree.
For $i = 1$ to $n - 2$
    Let v be the 1-valent vertex with the smallest label.
    Let s_i be the label of the only neighbor of v.
    $T := T - v$.
Return sequence $\langle s_1, s_2, \ldots, s_{n-2} \rangle$.

1



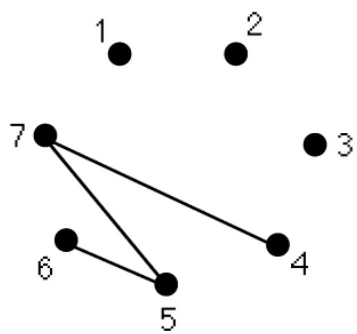
S = (7,

2



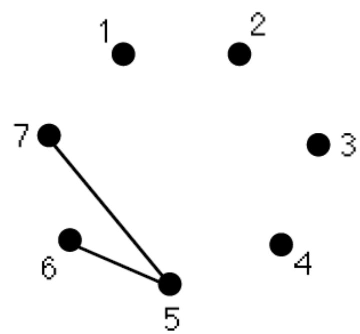
S = (7, 4,

3



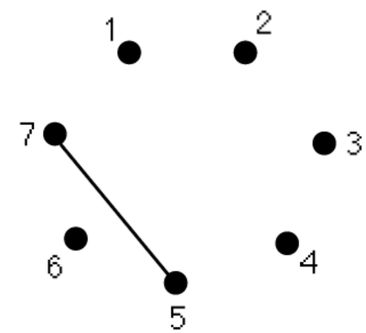
S = (7, 4, 4,

4



S = (7, 4, 4, 7,

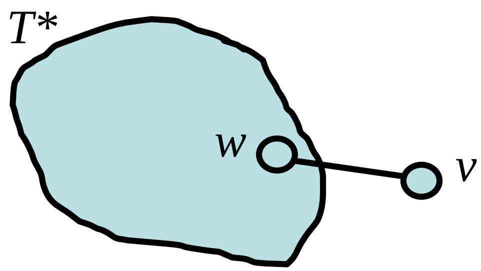
5



S = (7, 4, 4, 7, 5)

Prüfer Encoding

- **Proposition 3.7.1.** Let d_k be the number of occurrences of number k in a Prüfer encoding sequence for a labeled tree T. Then the degree of vertex k in T equals $d_k + 1$.
  - ✓ Prove by induction. The assertion is true for any tree on 3 vertices. Let T be a standard labeled tree on n+1 vertices and v be the 1-valent vertex with the smallest label and w be v's neighbor.
  - ✓ T^* is obtained by removing v from T. T^* is a standard labeled tree on n vertices, so $\forall k \in V_{T^*}$, $deg_{T^*}(k) = d_k(T^*) + 1$. $deg_T(w) = deg_{T^*}(w)+1$ and $d_w(T) = d_w(T^*)+1 \rightarrow deg_T(w) = d_w(T)+1$.



Prüfer Decoding

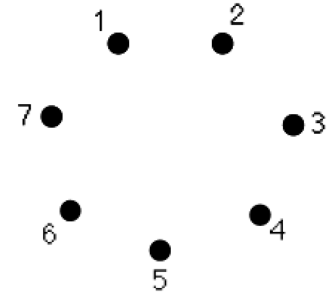
ALGORITHM: PRÜFER DECODING

Input: a Prüfer sequence of length $n - 2$.

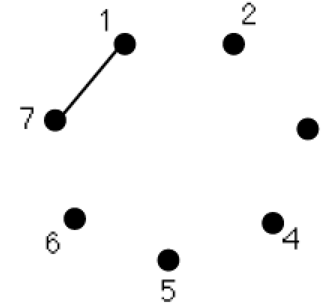
Output: an n-vertex tree with std 1-based vertex-labels.

Initialize list P as the Prüfer input sequence.
Initialize list L as $1, \ldots, n$.
Initialize forest F as n isolated vertices, labeled 1 to n.
For $i = 1$ to $n - 2$
  Let k be the smallest # in list L that is not in list P.
  Let j be the first number in list P.
  Add an edge joining the vertices labeled k and j.
  Remove k from list L.
  Remove the first occurrence of j from list P.
Add an edge joining the vertices labeled with the two remaining numbers in list L.
Return F with its vertex-labeling.

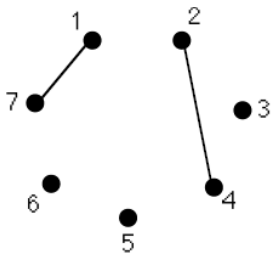
L = 1, 2, 3, 4, 5, 6, 7 P = 7, 4, 4, 7, 5



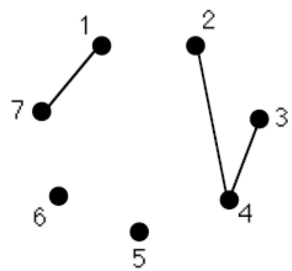
L = 2, 3, 4, 5, 6, 7 P = 4, 4, 7, 5



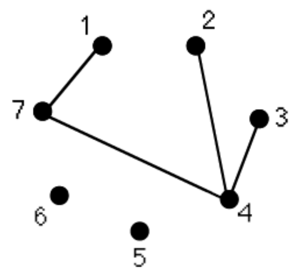
L = 3, 4, 5, 6, 7 P = 4, 7, 5



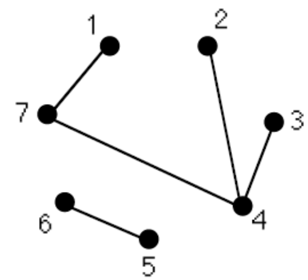
L = 4, 5, 6, 7 P = 7, 5

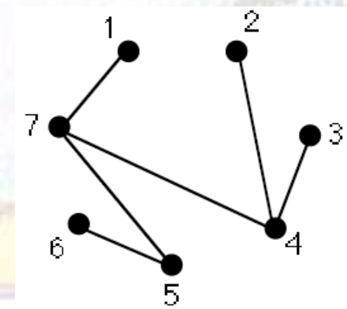


L = 5, 6, 7 P = 5



L = 5, 7





Prüfer Decoding

- **Proposition 3.7.2.** The decoding procedure defines a function $f_d : P_{n-2} \to T_n$ from the set of Prüfer sequences of length $n - 2$ to the set of labeled trees on n vertices.
  - ✓ Each step of decoding procedure specifically define a graph operation. We can be sure decoding procedure will create a graph. We have to prove the graph is a tree.
  - ✓ The assertion is true for $n = 2$ as the procedure produces a single edge. Assume the assertion is true for some $n \geq 2$. Consider a Prüfer sequence $(p_1, p_2, \ldots, p_{n-1})$ and a set of vertices $\{1, 2, \ldots, n+1\}$.
  - ✓ The first iteration of the procedure draws an edge from b to p_1, where b is the smallest vertex not appearing among the p_i's. None of the n-1 edges that are produced in iteration 2 through n will be incident with b. Thus continuing the procedure from iteration 2 is equivalent to applying the procedure to the Prüfer sequences $(p_2, \ldots, p_{n-1})$ for the set of vertices $\{1, \ldots, b-1, b+1, \ldots, n+1\}$. By the induction hypothesis, the edges produced form a tree on these vertices. This tree, together with the edge from b to p_1, forms a tree on the vertices $\{1, 2, \ldots, n+1\}$.
- **Proposition 3.7.3.** The decoding function $f_d : P_{n-2} \to T_n$ is the inverse of the encoding. function $f_e : T_n \to P_{n-2}$.

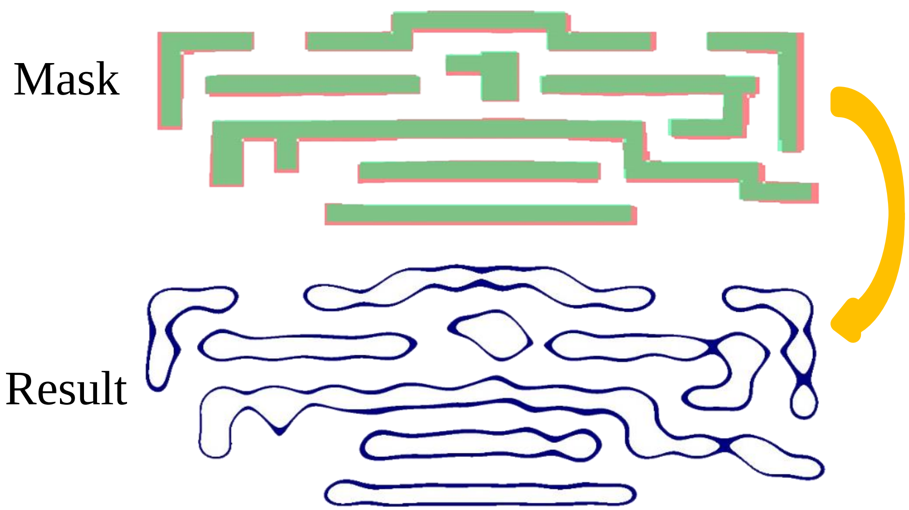
$$(1, 2, 3, \ldots, b, b+1, \ldots, n) \qquad (p_1, p_2, \ldots, p_{n-1})$$

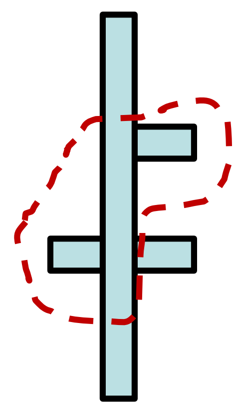
Prüfer Decoding

- **Theorem 3.7.4 [Cayley's Tree Formula].** The number of different trees on n labeled vertices is n^{n-2}.
  - ✓ Pro. 3.7.3 builds a one-to-one correspondence between T_n and P_{n-2}. $(k_1, \ldots, k_{n-2}) \rightarrow n^{n-2}$
- **Remark:** A slightly different view of Cayley's Tree Formula is that it gives us the number of different spanning trees of the complete graph K_n. The next chapter is devoted to spanning trees.

Pruffer Encoding Application

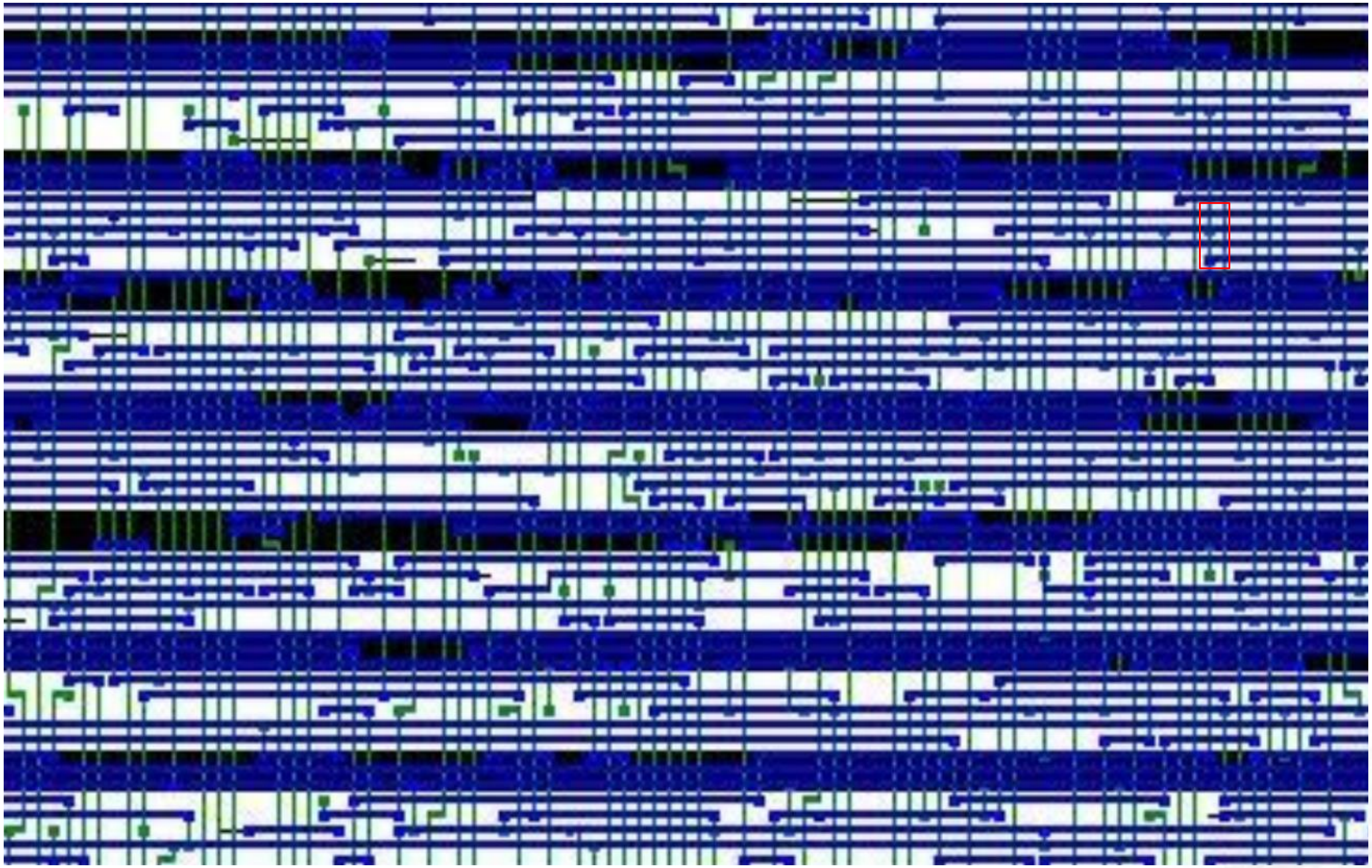
- Serious image distortion in advanced technology nodes – Design for Manufacturability (DFM) issues
- Pattern calibration – identify hot-spot patterns according to a set of pattern library
- Problem definition: given a layout possibly consisting of more than hundreds of million polygons and a set of 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.

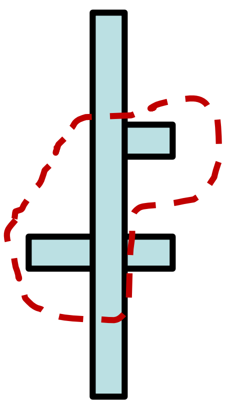


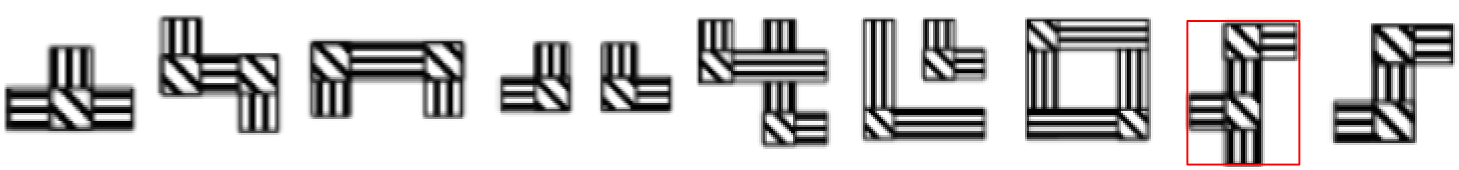


• Hong-Yan Su, Chieh-Chu Chen, Yih-Lang-Li, An-Chun Tu, Chuh-Jen Wu and Chen-Ming Huang, “A Novel Fast Layout Encoding Method for Exact Multi-Layer Pattern Matching with Prüfer-Encoding”, IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems, 2015.

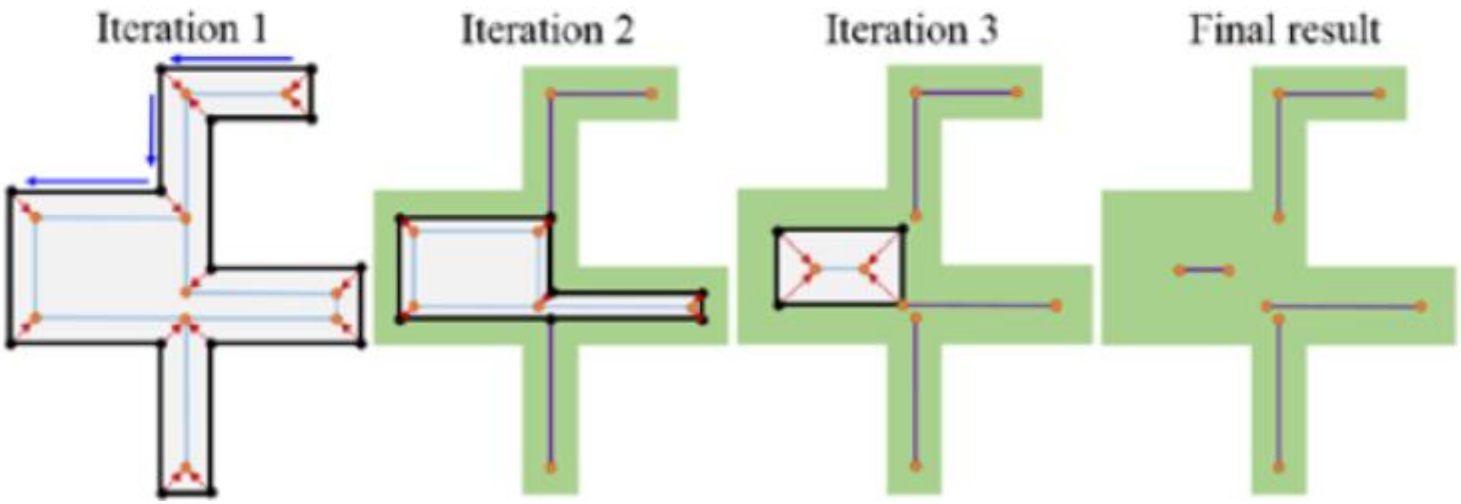
Pruffer Encoding Application



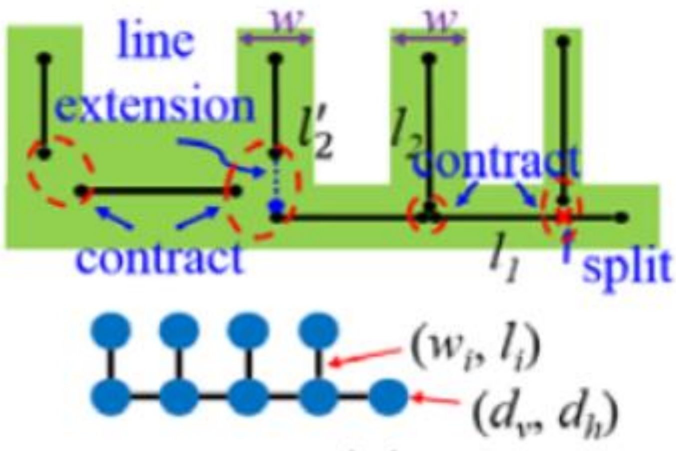




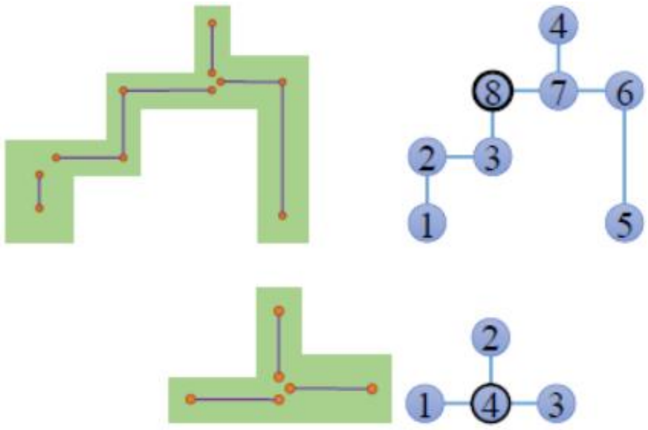
Pruffer Encoding Application



Polygon to centerlines

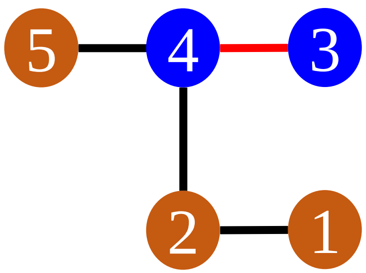


Centerlines to a tree

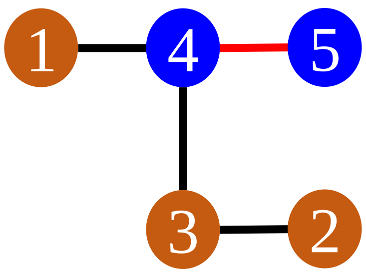


Partial recognition

EPC Transformation Algorithm



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



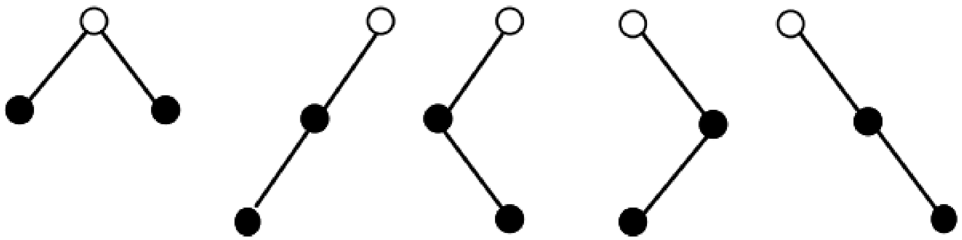
Q: How to find the function *f* ?

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**

Counting Binary Trees: Catalan Recursion

- Let b_n denote the number of binary trees on n vertices.
  - ✓ *Catalan Recursion*: $b_n = b_0 b_{n-1} + b_1 b_{n-2} + \ldots + b_{n-1} b_0$
  - ✓ b_n: the *nth Catalan number*.



- **Theorem 3.8.1.** *The number b_n of different binary trees on n vertices is given by* $b_n = \dfrac{1}{n+1}\dbinom{2n}{n}$.

$$b_n = b_0 b_{n-1} + b_1 b_{n-2} + \ldots + b_{n-1} b_0 = \sum_{i=0}^{n-1} b_i b_{n-1-i} ,$$

$$B(x) = b_0 + b_1 x + b_2 x^2 + \ldots = \sum_{n=0}^{\infty} \left(\sum_{i=0}^{n-1} b_i b_{n-1-i}\right) x^n \qquad (1), \quad A(x) = a_0 + a_1 x + a_2 x^2 + \ldots$$

$$C(x) = A(x)B(x) = c_0 + c_1 x + c_2 x^2 + \ldots = \sum_{n=0}^{\infty} c_n x^n, \quad c_n = a_0 b_n + a_1 b_{n-1} + \ldots + a_{n-1} b_1 + a_n b_0 = \sum_{i=0}^{n} a_i b_{n-i}$$

$$\text{now let } A(x) = B(x) \Rightarrow B(x)^2 = \sum_{n=0}^{\infty} \left(\sum_{i=0}^{n} b_i b_{n-i}\right) x^n, \text{ replace } n \text{ with } n-1 \Rightarrow B(x)^2 = \sum_{n=1}^{\infty} \left(\sum_{i=0}^{n-1} b_i b_{n-1-i}\right) x^{n-1}$$

$$\Rightarrow xB(x)^2 = \sum_{n=1}^{\infty} \left(\sum_{i=0}^{n-1} b_i b_{n-1-i}\right) x^n \Rightarrow \text{(By Eq. (1))} \ 1 + xB(x)^2 = B(x)$$

Counting Binary Trees: Catalan Recursion

$$B(x)=\frac{1\pm\sqrt{1-4x}}{2x}\Rightarrow B(x)=\frac{1-\sqrt{1-4x}}{2x}\text{ since }B(x)=\frac{1+\sqrt{1-4x}}{2x}=\frac{2}{0}\text{ as }x=0$$

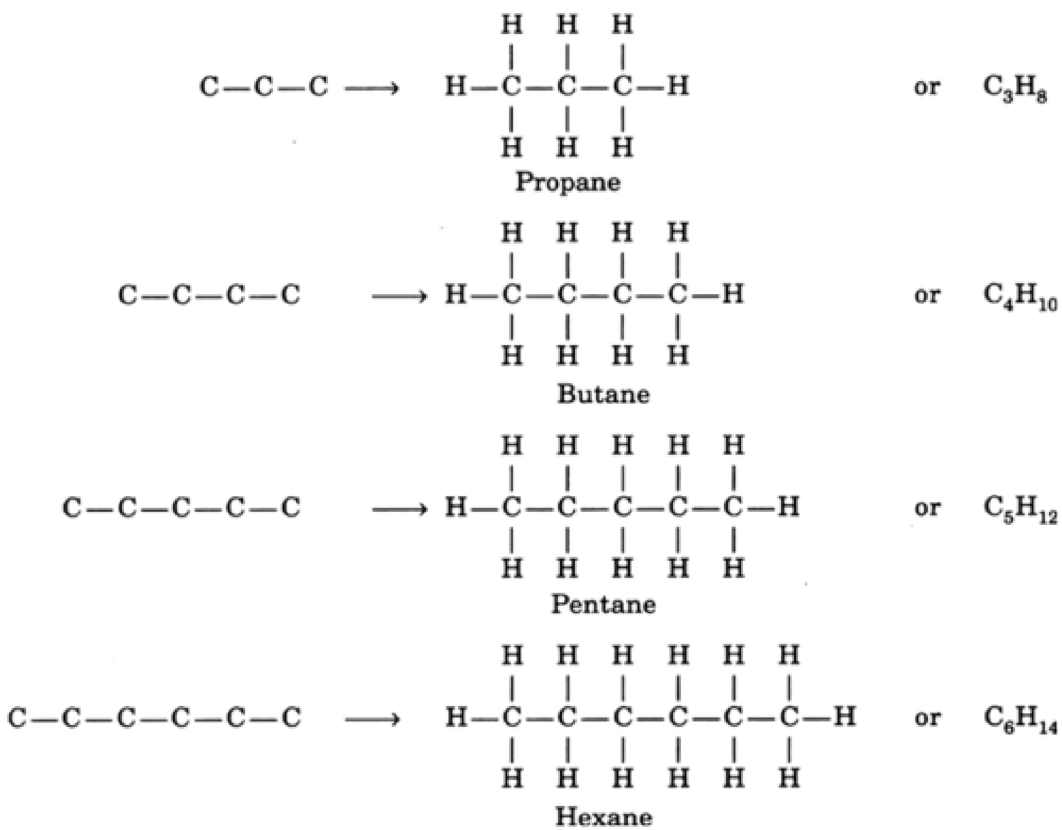
$$\because (x+y)^k=\sum_{n=0}^{\infty}\binom{k}{n}x^n y^{k-n}\ (\textit{Generalized Binomial Theorem})$$

$$\therefore (1-4x)^{\frac{1}{2}}=\sum_{n=0}^{\infty}\binom{\frac{1}{2}}{n}(-4x)^n\cdot 1^{1/2-n}=\sum_{n=0}^{\infty}\binom{\frac{1}{2}}{n}(-4x)^n$$

$$\therefore B(x)=\frac{1}{2x}\left[1-\sum_{n=0}^{\infty}\binom{\frac{1}{2}}{n}(-4x)^n\right]=\frac{1}{2x}\left[1-\sum_{n=1}^{\infty}\binom{\frac{1}{2}}{n}(-4x)^n-\binom{1/2}{0}(-4x)^0\right]$$

$$=\frac{-1}{2x}\sum_{n=1}^{\infty}\binom{\frac{1}{2}}{n}(-4x)^n=\sum_{n=1}^{\infty}\binom{\frac{1}{2}}{n}(-1)^{n+1}2^{2n-1}x^{n-1}=(\text{replace }n\text{ with }n+1)$$

$$\sum_{n=0}^{\infty}\binom{\frac{1}{2}}{n+1}(-1)^{n+2}2^{2(n+1)-1}x^n=\sum_{n=0}^{\infty}\binom{\frac{1}{2}}{n+1}(-1)^n 2^{2n+1}x^n\Rightarrow b_n=\binom{1/2}{n+1}(-1)^n 2^{2n+1}$$


C—C—C → or C_3H_8
Propane
C—C—C—C → or C_4H_{10}
Butane
C—C—C—C—C → or C_5H_{12}
Pentane
C—C—C—C—C—C → or C_6H_{14}
Hexane