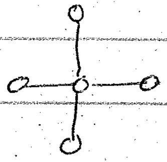
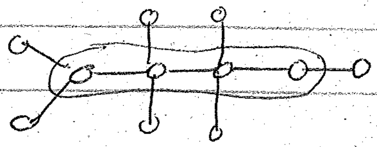
CHAPTER 3 – TREES & THEIR APPLICATIONS

§1. Trees, leaves, leaf-stems, and forests

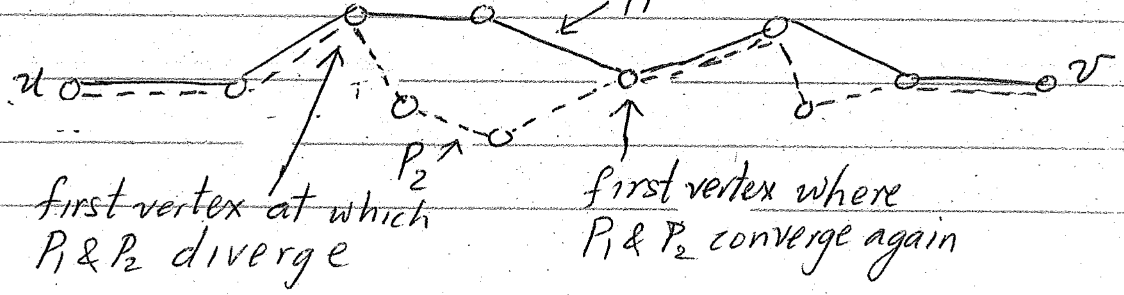
Def. A tree is a connected graph which contains no cycles. A leaf in a tree is a vertex of degree 1.

Ex.1 Let $T_1 =$  and $T_2 =$ 

Then T_1 & T_2 are trees. T_1 is called a star tree and has 4 leaves. T_2 is called a caterpillar tree because the non-leaf vertices all lie on a single path called the body of the caterpillar. A leaf-stem is the edge joing a leaf to the rest of the tree. The leaves & leaf stems are called the hairs of the caterpillar.

Prop. 1: A graph G is a tree $\Longleftrightarrow$ there is exactly one path between any two vertices of G.

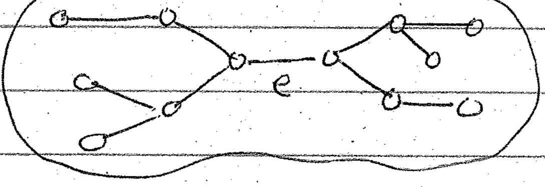
Proof: ($\Rightarrow$) Suppose G is a tree. Let u & v be any two vertices in G. Since G is connected there is at least one path from u to v in G. Now if there was more than one path from u to v in G, then we would get a cycle as shown below Hence there is only one path between u & v.

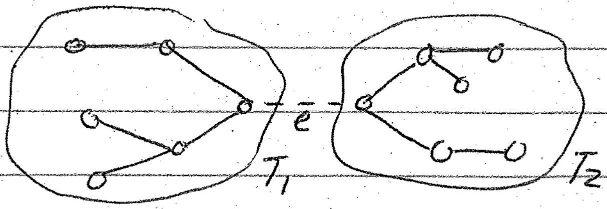

P_1
P_2
u
v
first vertex at which P_1 & P_2 diverge
first vertex where P_1 & P_2 converge again

($\Leftarrow$) Suppose there is there is exactly one path between any two vertices of G. Then G is connected (because there is a path between any two vertices of G). Now suppose G has a cycle with vertex sequence $v_1, v_2, v_3, \ldots, v_k, v_1$ say. Then we will get two paths from v_1 to v_k (namely $v_1, v_2, v_3, \ldots, v_k$ and v_1, v_k). So G cannot have any cycle. So G must be a tree.

Prop 2: Let T be any tree. Then $|E(T)| = |V(T)| - 1$.

Proof: Let $p = |V|$. We will prove the result by mathematical induction on p.
If T has 1 vertex, then T will be K_1 and so have 0 edges. Hence $|E(T)| = 0 = 1-1 = |V(T)| - 1$, So the result is for $p=1$.
Now suppose the result is true for all trees with $\leq p$ vertices. Then any tree with k vertices will have $k-1$ edges for $1 \leq k \leq p$. Let T be any tree with $p+1$ vertices and e be any edge in T. Then $T - \{e\}$ consists of two trees T_1 & T_2.

$T =$ 



$$\text{So } |E(T)| = |E(T_1)| + |E(T_2)| + |\{e\}| = \{V(T_1) - 1\} + \{V(T_2) - 1\} + 1$$
$$= |V(T_1)| + |V(T_2)| - 1 = |V(T)| - 1.$$

So if the result is true for all trees with $\leq p$ vertices, it will be true for all trees with $p+1$ vertices. Hence result is true for all trees.

Ex2 A certain tree T has 20 vertices of degree 5, 15 of degree 4, 10 of degree 3, and the rest of degree 1 or 2. How many leaves does T have? What is the minimum no. of vertices that T can have

Sol(a) Let p = no. of vertices in T, k = no. of vertices of degree 2 in T, and l = no. of leaves in T.

Then $20 + 15 + 10 + k + l = p$.

So $p = 45 + k + l$. Also

$$|E(T)| = |V(T)| - 1 = p - 1.$$

But sum of the degrees in $T = 2|E(T)|$. So

$$20(5) + 15(4) + 10(3) + k(2) + l(1) = 2(p-1)$$

$$\therefore \quad 100 + 60 + 30 + 2k + l = 2p - 2.$$

$$\therefore \quad 190 + 2k + l = 2(45 + k + l) - 2$$

$$\therefore \quad 190 + 2 - 90 = 2l - l$$

$\therefore \quad l = 102.$ So T has 102 leaves.

(b) From the equations above, we can see that the number of leaves does not depend on k. So if we set $k = 0$, we will get the minimum possible no. of vertices in T. This is

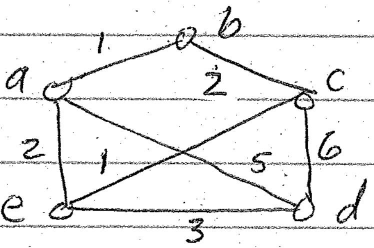
$$p_{min} = 45 + 0 + l = 45 + 102 = 147.$$

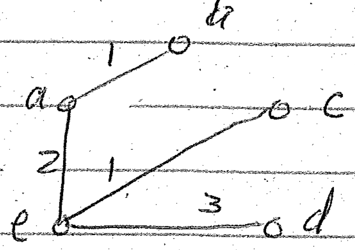
Def. A graph G that contains no cycles is called a <u>forest</u>. Each connected component of the forest G will be a tree, so the name is appropriate. A graph that contains exactly one cycle is called a unicyclic graph.

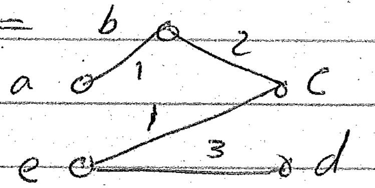
<u>Fact</u>: In any unicyclic graph G, $|E(G)| = |V(G)|$
<u>Proof</u>: Do for H.W.

§2. Minimum weight spanning trees

Def. Let $\langle G, w\rangle$ be a connected weighted graph. A spanning tree of G is any subgraph T of G such that T is a tree and $V(T) = V(G)$.
A minimum-weight spanning tree of G is any spanning tree T_0 of G such that $w(T_0) \le w(T)$ for any other spanning tree T of G,

Ex.1 Let $G =$ 

, $T_1 =$ 

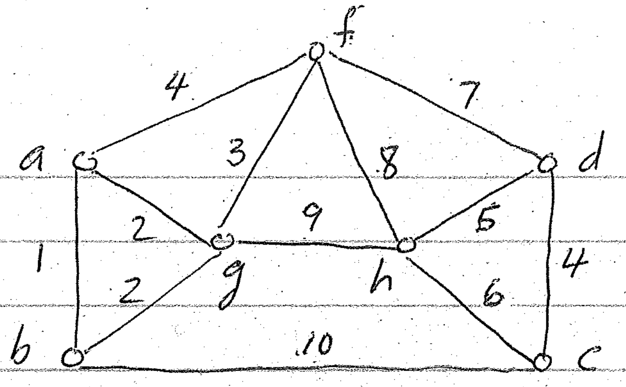
and $T_2 =$ 

. Then T_1 & T_2 are both minimum-weight spanning trees of G.

Algorithm 1 (Kruskal's Greedy Algorithm)
INPUT: A connected weighted graph G on $V = \{v_1, \dots, v_p\}$
OUTPUT: A minimum-weight spanning tree $T = \langle V, E(T)\rangle$ of G

```
1. Let $i \leftarrow 1$, $E(T) \leftarrow \emptyset$, and $P \leftarrow \{\{v_1\}, \{v_2\}, \dots, \{v_p\}\}$
   be taken as the initial partition with $p$ parts
2. Let $q = |E(G)|$ and list the edges of $G$ in increasing
   order of their weights to get a sequence $\langle e_1, \dots, e_q\rangle$ with
       $w(e_1) \le w(e_2) \le \dots \le w(e_q)$.  (split ties arbitrarily)
3. If the endpoints of $e_i$ are in different parts of $P$ (i.e,
   if $T \cup \{e_i\}$ contains no cycles) then let $E(T) \leftarrow E(T) \cup \{e_i\}$
   and unite the two parts from the which the endpoints of
   $e_i$ came to get a new $P$; else (if the endpoints of
   $e_i$ are from the same part of $P$) leave things unchanged.
4. Let $i \leftarrow i+1$. If $P$ has only one part, STOP; else go to step 3.
```

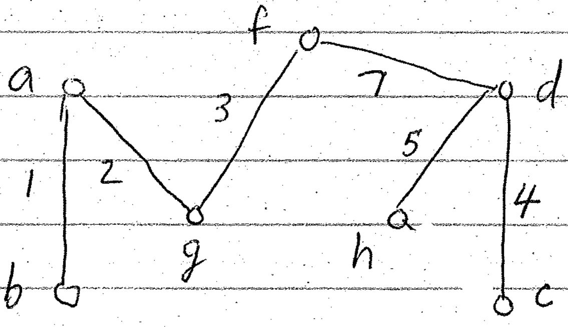
Ex.2 Let $G =$



Find a minimum-weight spanning tree of G by using Kruskal's Greedy Algorithm.

Sol. Let $e_1 = ab$, $e_2 = ag$, $e_3 = bg$, $e_4 = fg$, $e_5 = af$, $e_6 = cd$, $e_7 = dh$, $e_8 = ch$, $e_9 = df$, $e_{10} = fh$, $e_{11} = gh$, $e_{12} = bc$.

$E(T)$	Parts of $\mathcal{P}$	i	endpts(e_i)
$\emptyset$	$\{a\}\{b\}\{c\}\{d\}\{f\}\{g\}\{h\}$	1	$\{a,b\}$
$\{ab\}$	$\{a,b\}\{c\}\{d\}\{f\}\{g\}\{h\}$	2	$\{a,g\}$
$\{ab, ag\}$	$\{a,b,g\}\{c\}\{d\}\{f\}\{h\}$	3	$\{b,g\}$
$\{ab, ag\}$	$\{a,b,g\}\{c\}\{d\}\{f\}\{h\}$	4	$\{f,g\}$
$\{ab, ag, fg\}$	$\{a,b,f,g\}\{c\}\{d\}\{h\}$	5	$\{a,f\}$
$\{ab, ag, fg\}$	$\{a,b,f,g\}\{c\}\{d\}\{h\}$	6	$\{c,d\}$
$\{ab, ag, fg, cd\}$	$\{a,b,f,g\}\{c,d\}\{h\}$	7	$\{d,h\}$
$\{ab, ag, fg, cd, dh\}$	$\{a,b,f,g\}\{c,d,h\}$	8	$\{c,h\}$
$\{ab, ag, fg, cd, dh\}$	$\{a,b,f,g\}\{c,d,h\}$	9	$\{d,f\}$
$\{ab, ag, fg, cd, dh, df\}$	$\{a,b,c,d,f,g,h\}$ one part	STOP	



$w(T) = 1+2+3+4+5+7 = 22.$

<u>Algorithm 2</u> (Prim's closest-vertex Algorithm)
<u>INPUT</u>: A connected weighted graph $G = \langle V(G), E(G) \rangle$
<u>OUTPUT</u>: A minimum weight spanning tree $T = \langle V(T), E(T) \rangle$ of G.

1. Choose any vertex u_0 in $V(G)$. Let $E(T) = \emptyset$ & $V(T) \leftarrow \{u_0\}$. Also for each $x \in V(G) - V(T)$, let
$$d(x, V(T)) = \begin{cases} w(\overline{xu_0}) & \text{if } x \text{ is adjacent to } u_0 \text{ in } G \\ \infty & \text{if } x \text{ is not adjacent to } u_0 \text{ in } G. \end{cases}$$
2. Choose an edge $e = \overline{ux_0}$ from $V(T)$ to $V(G) - V(T)$ with $u \in V(T)$ and $x_0 \in V(G) - V(T)$ such that
$$d(x_0, V(T)) = \min\{ d(x, V(T)) : x \in V(G) - V(T)\}.$$
3. Let $E(T) \leftarrow E(T) \cup \{e\}$ and $V(T) \leftarrow V(T) \cup \{x_0\}$. If $V(T) = V(G)$ STOP.
4. For each $x \in V(G) - V(T)$ that is adjacent to x_0, let
$$d(x, V(T)) \leftarrow \min\{ w(\overline{x_0x}), d(x, V(T) - \{x_0\}) \}$$
Then go to step 2.

<u>Ex. 5</u> Find a minimum weight spanning tree T of the weighted graph in Ex. 4 by using Prim's Algorithm & start with $u_0 = c$.

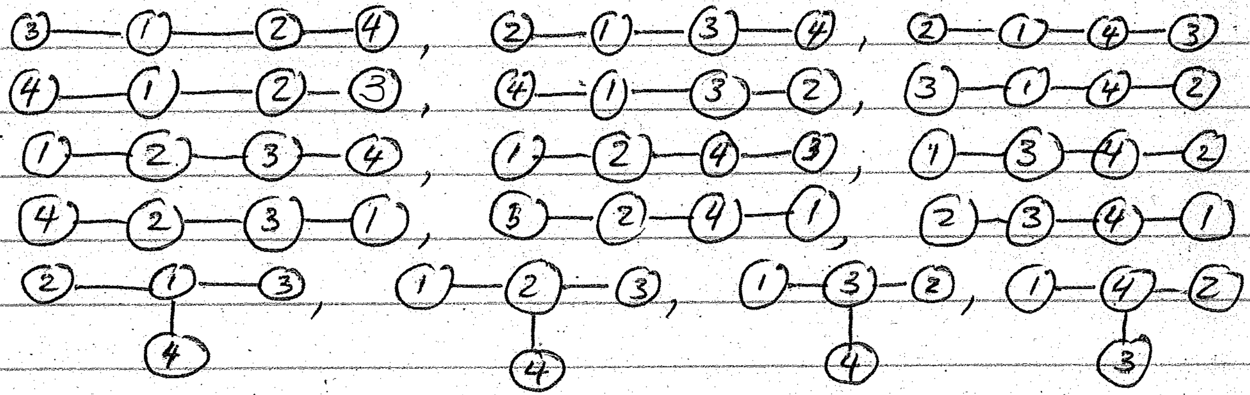
		$d(\cdot, V(T))$							
$E(T)$	$V(T)$	a	b	c	d	f	g	h	x_0
$\emptyset$	$\{c\}$	∞	10	.	$\underline{4}$	∞	∞	6	d
$\{cd\}$	$\{c, d\}$	∞	10	.	.	7	∞	$\underline{5}$	h
$\{cd, dh\}$	$\{c, d, h\}$	∞	10	.	.	$\underline{7}$	9	.	f
$\{cd, dh, df\}$	$\{c, d, f, h\}$	4	10	.	.	.	$\underline{3}$	.	g
$\{cd, dh, df, fg\}$	$\{c, d, f, g, h\}$	$\underline{2}$	2	.	.	.	.	.	a
$\{cd, dh, df, fg, ag\}$	$\{a, c, d, f, g, h\}$	.	$\underline{1}$	.	.	.	.	.	b
$\{cd, dh, df, fg, ag, ab\}$	$\{a, b, c, d, f, g, h\}$	STOP because $V(T) = V(G)$.							

§3 Counting non-identical trees on $V=\{1,2,\cdots,p\}$.

Our goal in this section is to prove Cayley's theorem which says that there are p^{p-2} non-identical trees on $V=\{1,2,\cdots,p\}$. Along the way, we would also get the most compact representation of trees on V by using the sequences in V^{p-2}. Unfortunately, there is no simple formula for the number of non-isomorphic trees on p vertices.

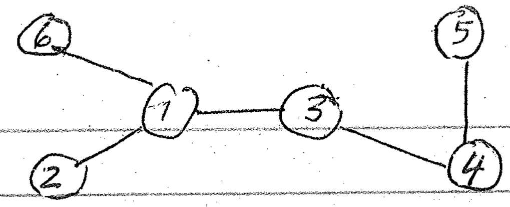
No. of vertices in V, p	1	2	3	4	5	6
No. of non-identical trees on V	1	1	3	16	125	1,296
No. of non-isomorphic trees on V	1	1	1	2	3	5

Let us look at the 16 non-identical trees on $V=\{1,2,3,4\}$

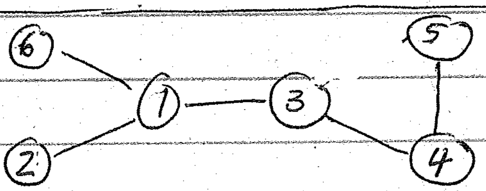
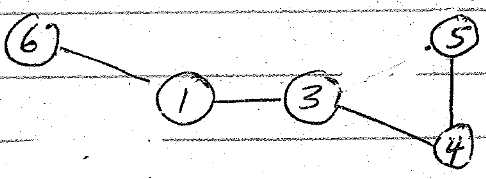
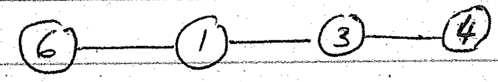
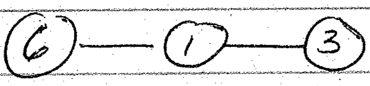
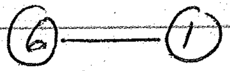


Algorithm 3:- (Prüfer's Coding Algorithm)

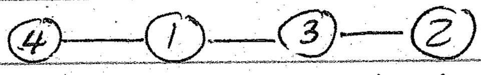
```
INPUT: A tree $T_0$ on the vertices $V=\{1,2,3,\ldots,p\}$
OUTPUT: A sequence $\underset{\sim}{s}$ of length $p-2$ of elements of $V$.
1. Let $i \leftarrow 1$ and $T \leftarrow T_0$.
2. Let $\ell(i) \leftarrow$ the leaf in $T$ with the smallest value, and
   $s(i) \leftarrow$ the unique vertex in $T$ to which vertex $\ell(i)$ is adjacent
3. Put $i \leftarrow i+1$ and $T \leftarrow T-\{\ell(i)\}$. If $i=p-1$, STOP; else go to step 2.
```

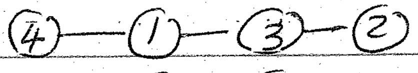
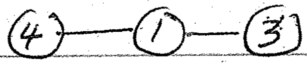
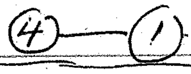
Ex. 1 Let T_0 =  and $V = \{1,2,3,4,5,6\}$

Find the sequence $\underset{\sim}{s}$ in $V \times V \times V \times V = V^4$ that codes T_0 via the Prüfer's Coding Algorithm. Here $p = 6$

i	T	$\ell(i)$	$s(i)$
1		2	1
2		5	4
3		4	3
4		3	1
5		STOP bec. $i = p-1$.	

Code of T_0 is $\underset{\sim}{s} = \langle s(1), s(2), s(3), s(4) \rangle = \langle 1, 4, 3, 1 \rangle$.

Ex. 2 Let T_0 =  and $V = \{1,2,3,4\}$.
Find the sequence $\underset{\sim}{s}$ in $V \times V = V^2$ that codes the tree T_0 via Prüfer's Coding Algorithm. Here $p = 4$.

i	T	$\ell(i)$	$s(i)$
1		2	3
2		3	1
3		STOP bec. $i = p-1$.	

So $\text{code}(T_0) = \langle s(1), s(2) \rangle = \langle 3, 1 \rangle$.

Algorithm 4 (Prüfer's Decoding Algorithm)

INPUT: A sequence $\underset{\sim}{s}$ of length $p-2$ of elements of $V=\{1,2,\cdots,p\}$.

OUTPUT: A tree T_0 on the vertices $\{1,2,3,\cdots,p\}=V$.

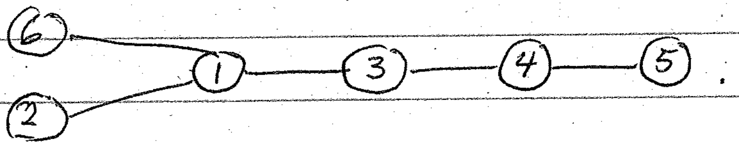
METHOD: Let $T(i)$ be the forest constructed by stage i. For each $x\in V$, let $d_i(x)$ be the amount of degrees vertex x still needs to make $T(i)$ into T_0.

1. Let $i \leftarrow 1$ & $T=\langle V,\emptyset\rangle$. For each vertex $x\in V$, let $d_i(x) = 1 +$ (no. of times x appears in the seq. $\underset{\sim}{S}$)
2. Let $l(i) \leftarrow$ smallest vertex with $d_i(l(i))=1$. Put $T\leftarrow T\cup\{l(i)-s(i)\}$, $d_{i+1}[l(i)]\leftarrow 0$, $d_{i+1}[s(i)]\leftarrow d_i[s(i)]-1$, and $i\leftarrow i+1$.
3. If $i<p-1$, go to step 2; else (if $i=p-1$), let $T\leftarrow T\cup\{e\}$, where is the edge joining the last two vertices which need 1 degree each, and STOP.

Ex. 3 Find the tree T_0 that is coded by the sequence $\underset{\sim}{s} = \langle 1,4,3,1\rangle$ via Prüfer's Decoding Algorithm.

Sol. Here $p-2=|\underset{\sim}{s}|=4$. So $p=6$ and thus $V=\{1,2,3,\cdots,6\}$.

$d_i(1)$	$d_i(2)$	$d_i(3)$	$d_i(4)$	$d_i(5)$	$d_i(6)$	i	$l(i)$	$s(i)$
3	1	2	2	1	1	1	2 —	1
2	0	2	2	1	1	2	5 —	4
2	0	2	1	0	1	3	4 —	3
2	0	1	0	0	1	4	3 —	1
1	0	0	0	0	1	5	1 —	6 STOP

$T_0 =$  .

Ex. 4 Find the tree T_0 that is coded by the sequence $\underset{\sim}{s} = \langle 3, 1 \rangle$ via the Prüfer's Decoding Algorithm.

Sol. Here $p-2 = |\underset{\sim}{s}| = 2$. So $p = 4$ & thus $V = \{1, 2, 3, 4\}$

$d_i(1)$	$d_i(2)$	$d_i(3)$	$d_i(4)$	i	$l(i)$	$s(i)$
2	$\underline{1}$	$\underline{2}$	1	1	2 —	3
$\underline{2}$	0	$\underline{1}$	1	2	3 —	1
1	0	0	1	3	1 —	4 STOP.

$$\therefore \quad T_0 = \textcircled{4} - \textcircled{1} - \textcircled{3} - \textcircled{2}.$$

Theorem 3 (Cayley's Tree Theorem): The number of non-identical trees on $V = \{1, 2, \dots, p\}$ is p^{p-2}.

Proof: Let $\mathcal{T}(V)$ set of all trees on V. We will show that there is a bijection $f: \mathcal{T}(V) \to V^{p-2}$. Here $V^{p-2} = V \times V \times \dots \times V$ ($p-2$ times) $=$ set of all seq. of length $p-2$ of elements of V. From this it will follow that

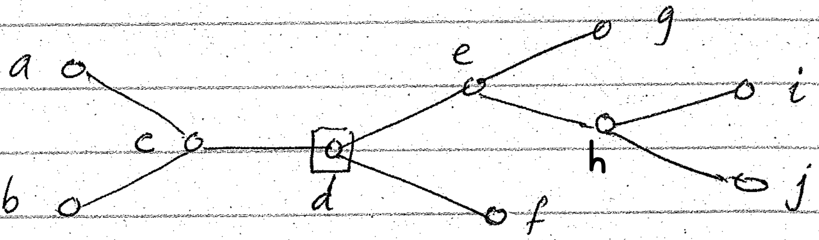
$$|\mathcal{T}(V)| = |V^{p-2}| = (|V|)^{p-2} = p^{p-2}.$$

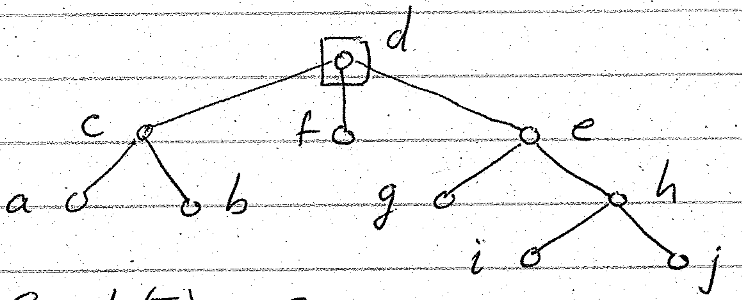
We define $f: \mathcal{T}(V) \to V^{p-2}$ by $f(T) =$ the sequence of length $p-2$ of elements of V obtained from Prüfer's Coding Algorithm. Now define the function $g: V^{p-2} \to \mathcal{T}(V)$ by $f(\underset{\sim}{s}) =$ the tree on V obtained from Prüfer's Decoding Algorithm. It is easy to see that $g(f(T)) = T$ for each $T \in \mathcal{T}(V)$ and that $f(g(\underset{\sim}{s})) = \underset{\sim}{s}$ for each $\underset{\sim}{s} \in V^{p-2}$. From this it follows that $g = f^{-1}$. Hence f is a bijection and the proof is now complete.

§4. Rooted trees & optimal binary codings

Def. A rooted tree is an ordered pair $\langle T, v_0\rangle$ where T is a tree and v_0 is a distinguished vertex in T. The vertex v_0 is called the root of the rooted tree. The vertices of a rooted tree $\langle T, v_0\rangle$ can be classified into levels according to their distances from the root v_0. The height $h(T)$ of a rooted tree is the highest level that exists in the rooted tree.

Ex.1 Let $T =$





— level 0
— level 1
— level 2
— level 3.

So $h(T) = 3$.

Def. If a vertex v at level k is adjacent to the vertices $u_1, u_2, \dots, u_n$ at level $k+1$, then we say that v is the parent of $u_1, \dots, u_n$; and that $u_1, \dots, u_n$ are the children of v.

Ex.2 In the tree in Ex.1, e is the parent of g & h; and g & h are children of e. The vertex g has no children and is said to be childless. Note that the childless vertices will be leaves in any non-trivial tree.

Def. An <u>n-ary tree</u> is a rooted tree in which each vertex has at most n children. The n-ary tree is said to be <u>full</u> if every vertex has exactly n children or no children. The n-ary tree T is <u>balanced</u> if every childless vertex is at level $h(T)$ or level $h(T)-1$.

Prop. 4 For any r-ary tree T with p vertices we have
$$\log_r\left[\frac{p(r-1)+1}{r}\right] \leq h(T) \leq p-1.$$

Proof: Let $k = h(T)$. Since each level of the tree must have at least one vertex, it follows there can be at most p levels in T. Since we start from level 0, the highest level in T cannot be more than $p-1$. So $k \leq p-1$. So $h(T) \leq p-1$.

Also since T is an r-ary tree, level 0 can contain at most r^0 vertices; level 1, at most r^1 vertices; and so on. In general level i can contain at most r^i vertices. So
$$p \leq r^0 + r^1 + r^2 + \cdots + r^k = \frac{r^{k+1}-1}{r-1}$$

$\therefore\ p(r-1) \leq r^{k+1}-1$. So $p(r-1)+1 \leq r.r^k$

Thus $r^k \geq [p(r-1)+1]/r$. Hence
$$\log_r(r^k) \geq \log_r\left[\frac{p(r-1)+1}{r}\right].$$
So $h(T) = k \geq \log_r\left[\frac{p(r-1)+1}{r}\right]$. Thus
$$\log_r\left[\frac{p(r-1)+1}{r}\right] \leq h(T) \leq p-1.$$

Ex. 3 If T is a binary tree with 35 vertices, then
$\log_2\left[\frac{35(2-1)+1}{2}\right] \leq h(T) \leq 35-1$. So $\log_2(18) \leq h(T) \leq 34$
$\therefore\ 5 \leq h(T) \leq 34$.

<u>Def.</u> An <u>ordered $_\wedge$ tree</u> is a rooted tree in which the children of each vertex are listed in some linear order from left to right.

rooted

<u>Def.</u> Let $a_1, a_2, \cdots, a_k$ be k characters. A <u>binary coding</u> of $a_1, \cdots, a_k$ is a function $c: \{a_1, \cdots, a_k\} \to \{0,1\}^*$. Here $\{0,1\}^*$ is the of all strings of 0's & 1's.
$\{0,1\}^* = \{\lambda, 0, 1, 00, 01, 10, 00, 000, 001, \cdots\}$
A binary coding is said to be <u>uniquely decipherable</u> if every string in $\{0,1\}^*$ represents at most one string of characters from $\{a_1, a_2, \cdots, a_k\}$.

<u>Ex.4</u> Let $c: \{x, y, z\} \to \{0,1\}^*$ be defined by $c(x) = 10$, $c(y) = 11$ & $c(z) = 0$. Then c is a uniquely decipherable coding of $\{x, y, z\}$. The string 110100 codes $yzxz$.

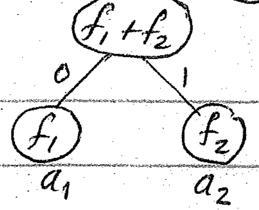
<u>Def.</u> Let c be a coding of the characters $\{a_1, \cdots, a_k\}$. The <u>length</u> of the code $c(a_i)$ of a_i defined to be the length of the string $c(a_i)$. If the character a_i occurs with percentage frequencies $f(a_i)$ we define the <u>weighted path length</u> of the coding c by

$$WPL(c) = \sum_{i=1}^{k} f(a_i) . \text{length}[c(a_i)]$$

A uniquely decipherable coding c_0 is said to be <u>optimal</u> if $WPL(c_0) \leq WPL(c)$ for any other uniquely decipherable coding c of $\{a_1, \cdots, a_k\}$.

<u>Algorithm 5</u> (Huffman's Optimal Coding Algorithm)
<u>INPUT</u>: A set of k characters & their corresponding % frequencies
<u>OUTPUT</u>: An optimal binary coding for the k characters.

1. List the frequencies in increasing order
2. Take the two smallest frequencies, say f_1 & f_2 of a_1 & a_2, and join them as on the right
3. Delete f_1 & f_2 from the list and add a new term f_1+f_2. Then reorder the list in increasing order
4. If the list has more than one term, go to step 2; else, go to step 5.
5. The code of each character is obtained by starting at the root of the coding tree and listing the sequence of 0's & 1's to the character in question. (Note that the original frequencies will be leaves in the coding tree and these vertices will be associated with characters.)



Ex 5 Find an optimal binary coding of the characters below.

Character	a	b	c	d	e	f
Frequencies	25	4	15	6	40	10

Sol.

$\underbrace{4, 6}, 10, 15, 25, 40$

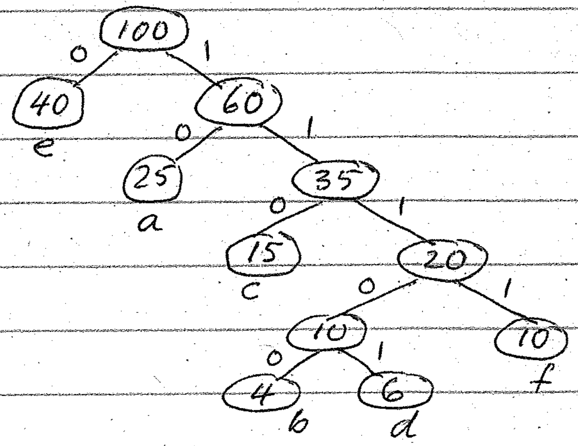
$\underbrace{10, 10}, 15, 25, 40$

$\underbrace{15, 20}, 25, 40$

$\underbrace{25, 35}, 40$

$\underbrace{40, 60}$

100.



Characters	a	b	c	d	e	f
Code	10	11100	110	11101	0	1111
length of code	2	5	3	5	1	4

$$\text{WPL (coding)} = 25(2) + 4(5) + 15(3) + 6(5) + 40(1) + 10(4)$$
$$= 50 + 20 + 45 + 30 + 40 + 40 = 225.$$

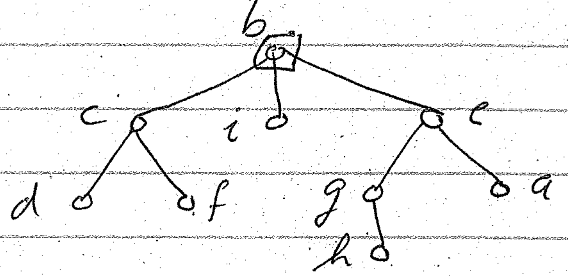
END

SKIP

~~§5.~~ Tree traversals & parentheses-free notations

Recall that an ordered rooted tree was a rooted tree in which the children of each vertex are linearly ordered. We usually list the vertices in decreasing order as in a family tree.

Ex.1 Consider the rooted tree T, on the right. $T =$

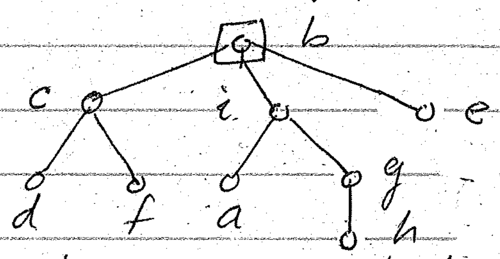


We can make T into an ordered rooted tree by letting

$c > i > e$, $d > f$, and $g > a$

This will the depiction shown above.

If we let $c > e > i$, $d > f$ and $a > g$, then we we will get a slightly different ordered rooted tree T'.



Now T' is isomorphic to T as rooted trees but not as ordered rooted trees.

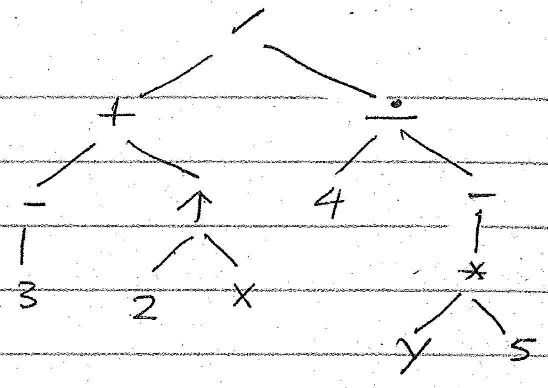
We can use ordered rooted trees to obtain parentheses-free notations for mathematical expressions.

Ex.2 Consider the expression

$$((-3) + (2 \uparrow x)) \;/\; 4 \div (-(y * 5))$$

Here "$-$" is the unary negative sign, $\div$ is the binary subtraction sign and $\uparrow, *, /$ are the exponentiation, multiplication and division signs.

Ex.2 From the expression we can obtain the expression tree shown on the right.



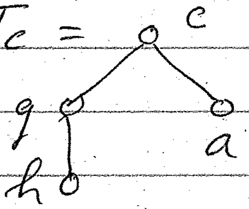
If we traverse the expression tree in pre-order traversal we will get the prefix notation for the expression.

We get the post-fix & in-fix notations by traversing the expression tree in post-order and in-order, respectively.

Def. Let T be a rooted tree. We say that w is a descendant of u if we can find a sequence of vertices $u = v_1, v_2, v_3, \ldots, v_n = w$ such that v_{i+1} is a child of v_i for each $i = 1, \ldots, n-1$. We say that w is an ancestor of u if u is a descendant of w.

Def Let T be an ordered rooted tree. If v is a vertex in T, we define T_v to be the sub-tree of T which consists of v and all its descendants.

Ex.3 In the ordered rooted tree T from Ex.1, a, g and h are descendants of c. Also g, c and b are ancestors of h. The sub-tree T_c is shown on the right, above.

$T_c =$


Def We define the preorder traversal of an ordered rooted tree as the string of vertices, pre(T), that is defined recursively as follows.

(a) If T has no children (i.e., if T consists of a single vertex), we define pre(T) = root(T).

(b) If the children of T are $u_1, u_2, \dots, u_k$ (ordered from left to right), then we define
$$pre(T) = root(T)\, pre(T_{u_1})\, pre(T_{u_2}) \cdots pre(T_{u_k}).$$

The post order traversal of T will also be defined similarly by recursion.

Def (a) If T has no children, then post(T) = root(T)

(b) If the children of T are $u_1, u_2, \dots, u_k$ (ordered from left to right), then
$$post(T) = post(T_{u_1})\, post(T_{u_2}) \cdots post(T_{u_k})\, root(T).$$

Def. The Inorder traversal is defined only for ordered rooted binary trees.

(a) If T has no children, then In(T) = root(T).

(b) If T hase one child u, then $In(T) = In(T_u)\, root(T)$

(c) If T has two children u, v (ordered from left to right) then $In(T) = In(T_u)\, root(T)\, In(v)$.

Ex.4 Let us find the prefix notation for the expression tree of example 2.

(a) $pre(T) = /\ pre(T_+)\, pre(T_\div)$
$= /\ +\ pre(T_-)\, pre(T_\uparrow) \div pre(T_4)\, pre(T_-)$

$= / + - \text{pre}(T_3) \uparrow \text{pre}(T_2)\,\text{pre}(T_x) \div 4 - \text{pre}(T_*)$
$= / + - 3 \uparrow 2\ x \div 4 - * \ \text{pre}(T_y)\ \text{pre}(T_5)$
$= / + - 3 \uparrow 2\ x \div 4 - * \ y\ 5$

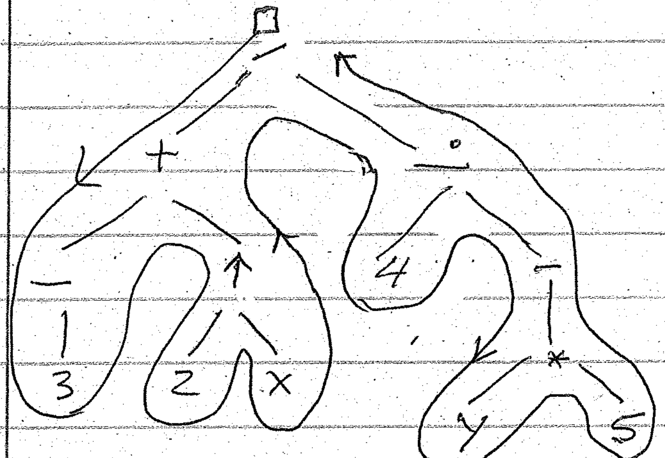
We can similarly the postfix & infix notations for T.

(b) $\text{post}(T) = 3 - 2\ x \uparrow + 4\ y\ 5 * - \div /$

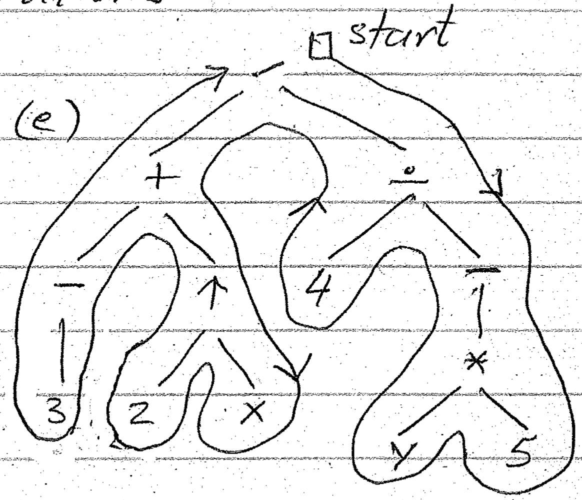
(c) $\text{Infix}(T) = 3 - + 2 \uparrow x / 4 \div y * 5 -$

There are, of course faster ways to visually find the prefix & postfix notations

(d)



(e)

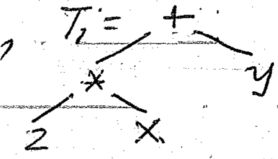


follow the path and write the sequence of vertices from left to right without repetitions.

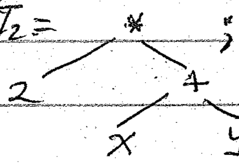
follow the path & write the seq. of vertices from right to left.

An expression, involving only unary & binary operations can always be recovered from its prefix or postfix notation. For infix notation, the expression cannot always be recovered because two different expressions can have the same infix notation.

$(2*x)+y$, $T_1 =$



; $2*(x+y)$, $T_2 =$



; infix notation $= 2 * x + y$ for both,

Below we will show how to recover the expressions from the prefix and postfix notations.

Recovery algorithm for Prefix notation:

1. Starting from the right end of the string, look for the first substring of the form
   (unary operation) (quantity) or
   (binary operation) (quantity) (quantity)
2. Use parentheses to make this substring a single quantity. If the new string contains a single quantity, STOP; otherwise go to step 1.

Ex. 5

$$/ \quad + \quad - \quad 3 \quad \uparrow \quad 2 \quad x \quad \div \quad 4 \quad - \quad \underbrace{* \quad Y \quad 5}$$

$$/ \quad + \quad - \quad 3 \quad \uparrow \quad 2 \quad x \quad \div \quad 4 \quad \underbrace{- \quad (Y*5)}$$

$$/ \quad + \quad - \quad 3 \quad \uparrow \quad 2 \quad x \quad \underbrace{\div \quad 4 \quad [-(Y*5)]}$$

$$/ \quad + \quad - \quad 3 \quad \underbrace{\uparrow \quad 2 \quad x} \quad \{4 \div [-(Y*5)]\}$$

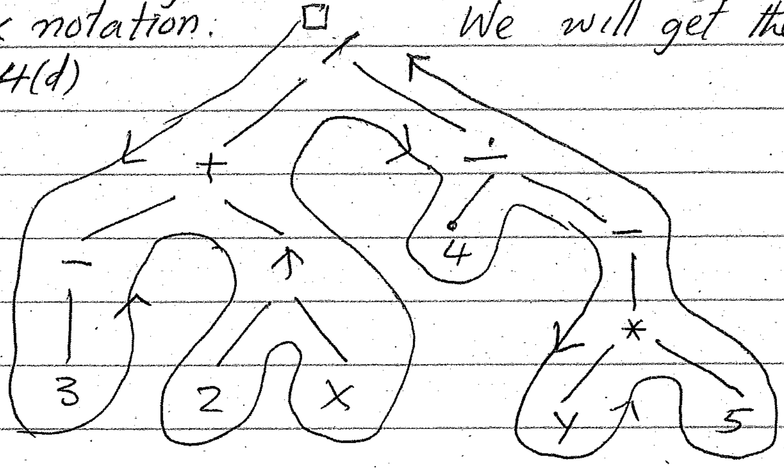
$$/ \quad + \quad \underbrace{- \quad 3} \quad (2 \uparrow x) \quad \{4 \div [-(Y*5)]\}$$

$$/ \quad \underbrace{+ \quad (-3) \quad (2 \uparrow x)} \quad \{4 \div [-(Y*5)]\}$$

$$\underbrace{/ \ \{(-3) + (2 \uparrow x)\} \ \{4 \div [-(Y*5)]\}}$$

$$\{(-3) + (2 \uparrow x)\} / \{4 \div [-(Y*5)]\}$$

We can also do this much faster by recovering the expression tree, T. Just branch with each binary operation ~~and follow the diagram for the short method of finding~~ the postfix notation. We will get the same tree as in Ex. 4(d)



Recovery algorithm for Postfix notation

1. Starting from the left end of the string, look for the first substring of the form
   (quantity) (unary operation)
   (quantity)(quantity) (binary operation)
2. Use parentheses to make this substring a single quantity. If the new string contains a single quantity, STOP; else go to step 1

Ex. 6

$$\underbrace{3\ -}\quad 2\quad x\quad \uparrow\quad +\quad 4\quad y\quad 5\quad *\quad -\quad \div\quad /$$

$$(-3)\quad \underbrace{2\quad x\quad \uparrow}\quad +\quad 4\quad y\quad 5\quad *\quad -\quad \div\quad /$$

$$\underbrace{(-3)\quad (2\uparrow x)\quad +}\quad 4\quad y\quad 5\quad *\quad -\quad \div\quad /$$

$$\{(-3)+(2\uparrow x)\}\quad 4\quad \underbrace{y\quad 5\quad *}\quad -\quad \div\quad /$$

$$\{(-3)+(2\uparrow x)\}\quad 4\quad \underbrace{(y*5)\quad -}\quad \div\quad /$$

$$\{(-3)+(2\uparrow x)\}\quad \underbrace{4\quad [-(y*5)]\quad \div}\quad /$$

$$\underbrace{\{(-3)+(2\uparrow x)\}\quad \{4\div[-(y*5)]\}\quad /}$$

$$\{(-3)+(2\uparrow x)\}/\{4\div[-(y*5)]\}$$

We can also do this much faster by recovering the expression tree T, as we did with the prefix notation. In this we start at the right end of the postfix notation and lay out the vertices and branch out two ways with the binary operations and downwards only with the unary operations. We will end up getting the same expression tree in Example 4(e)

END