

1. Give the pre-order traversal.
2. Give the post-order traversal.

procedure preorder(T: ordered rooted tree)

r := root of T

list r

for each child *c* of *r* from left to right

T(*c*) := subtree with *c* as root

preorder(T(*c*))

procedure postorder(T: ordered rooted tree)

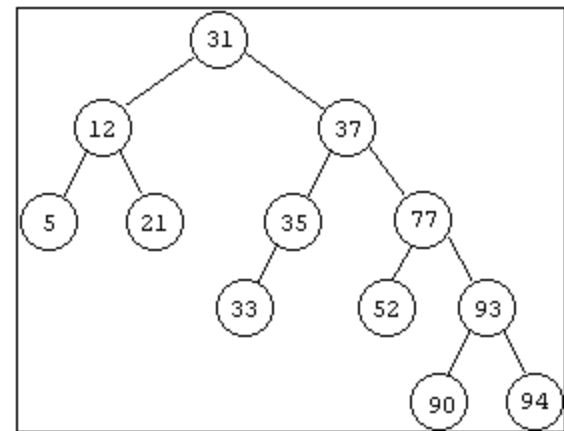
r := root of T

for each child *c* of *r* from left to right

T(*c*) := subtree with *c* as root

postorder(T(*c*))

list r



3. Give the in-order traversal.

procedure inorder(T: ordered rooted tree)

$r :=$ root of T

if r is leaf **then** list r

else

$l :=$ first child of r from left to right

$T(l) :=$ subtree with l as its root

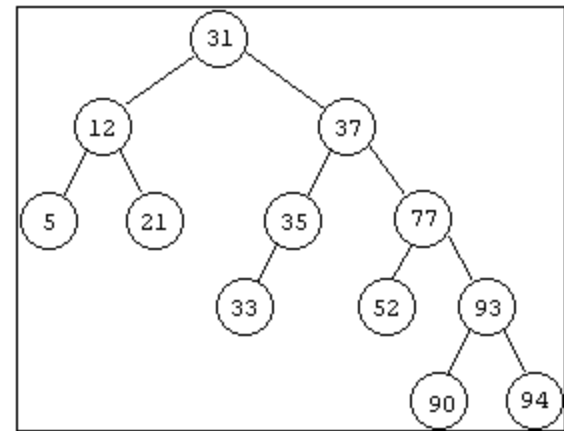
inorder($T(l)$)

list r

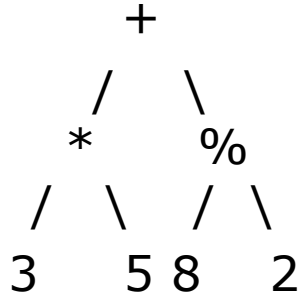
for each child c of r except for l from left to right

$T(c) :=$ subtree with c as root

inorder($T(c)$)



Binary tree representing: $3 * 5 + 8 \% 2$



4. Give the post-order traversal.

procedure postorder(T: ordered rooted tree)

 r := root of T

for each child *c* of *r* from left to right

 T(*c*) := subtree with *c* as root

 postorder(T(*c*))

 list r

5. Evaluate RPN: 5 4 * 9 3 + - 2 +

1. Scan RPN input from left to right.

1 | 2 | 3 | 4 input
push input onto stack.

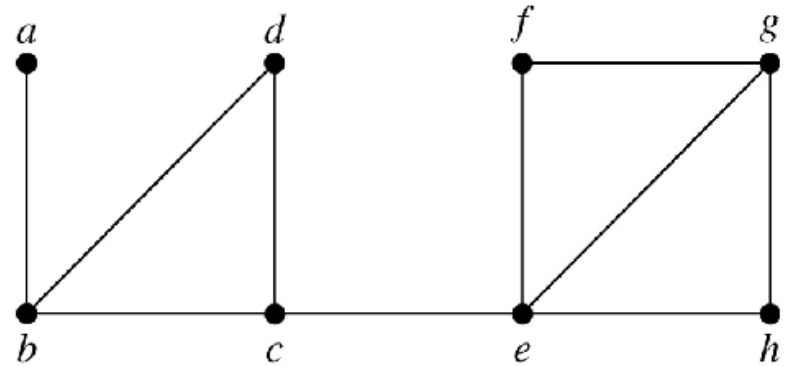
+ | - | * input

1. pop operand2
pop operand1
2. push operand1 * operand2
or
push operand1 - operand2
or
push operand1 + operand2

2. After RPN is scanned, the expression value is stack top.

Input	Stack	Push
<u>1</u> 2 3 * - 4 +	<u>1</u>	1
<u>2</u> 3 * - 4 +	1 <u>2</u>	2
<u>3</u> * - 4 +	1 2 <u>3</u>	3
* - 4 +	1 <u>6</u>	2 * 3
- 4 +	<u>-5</u>	1 - 6
<u>4</u> +	-5 <u>4</u>	4
±	<u>-1</u>	-5 + 4

6. Start at c , depth-first search for spanning tree.
Use alphabetically smaller letter first.



procedure DFS(G : connected graph, vertices: v_1, v_2, \dots, v_n)

$T :=$ tree consisting of v_1

visit(v_1)

procedure visit(v : vertex of G)

for each vertex w adjacent to v and not yet in T

add vertex w and edge (v, w) to T

visit(w)

7. Start at c , breadth-first search for spanning tree.
Use alphabetically smaller letter first.

procedure BFS(G : graph, vertices: v_1, v_2, \dots, v_n)

T := tree consisting of v_1

L := empty list

put v_1 in the list L of unprocessed vertices

while L is not empty

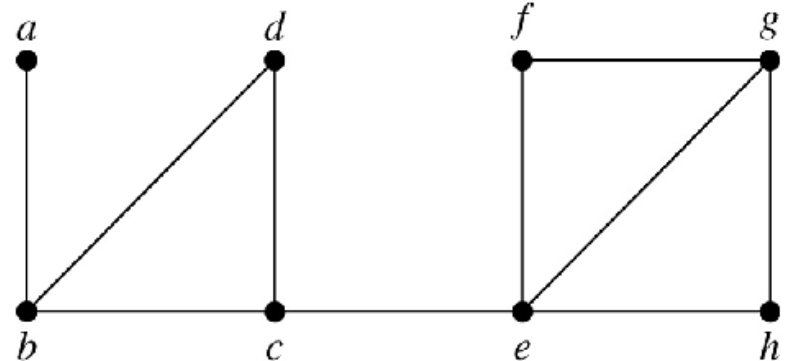
 remove the first vertex, v , from L

for each neighbor w of v

if w not in L and not in T **then**

 add w to the end of list L

 add w and edge (v, w) to T



1. Give the pre-order traversal.

31,12,5,21,37,35,33,77,52,93,90,94

2. Give the post-order traversal.

5,21,12,33,35,52,90,94,93,77,37,31

procedure preorder(T: ordered rooted tree)

 r := root of T

 list r

for each child *c* of *r* from left to right

 T(*c*) := subtree with *c* as root

 preorder(T(*c*))

procedure postorder(T: ordered rooted tree)

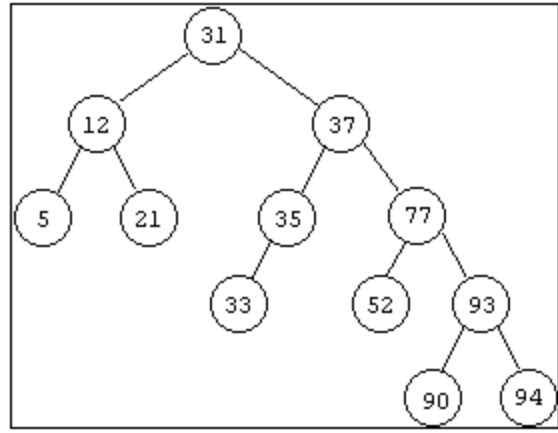
 r := root of T

for each child *c* of *r* from left to right

 T(*c*) := subtree with *c* as root

 postorder(T(*c*))

 list r



3. Give the in-order traversal.

5,12,21,31,33,35,37,52,77,90,93,94

procedure inorder(T: ordered rooted tree)

 r := root of T

if r is leaf **then** list r

else

 l := first child of r from left to right

 T(l) := subtree with l as its root

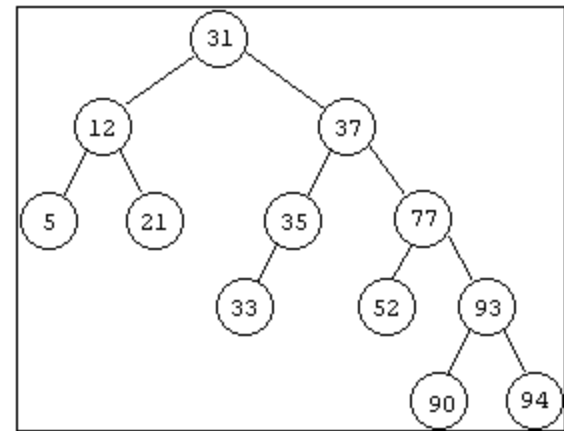
 inorder(T(l))

 list r

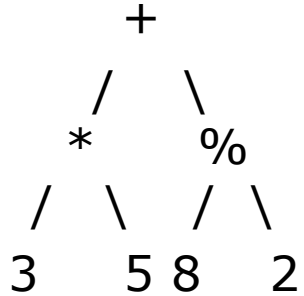
for each child c of r except for l from left to right

 T(c) := subtree with c as root

 inorder(T(c))



Binary tree representing: $3 * 5 + 8 \% 2$.



4. Give the post-order traversal. $3\ 5\ * \ 8\ 2\ \% \ +$

5. Evaluate: $5\ 4\ * \ 9\ 3\ + \ - \ 2\ + \ = \ 10$

procedure postorder(T: ordered rooted tree)

r := root of T

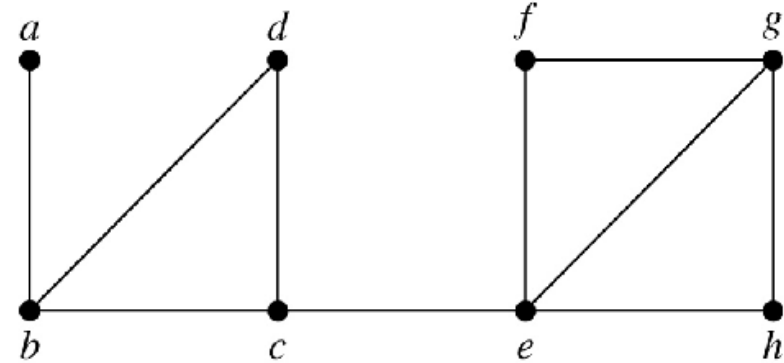
for each child *c* of *r* from left to right

 T(*c*) := subtree with *c* as root

 postorder(T(*c*))

list *r*

6. Start at c , depth-first search for spanning tree.
 Use alphabetically smaller letter first.



procedure DFS(G : connected graph, vertices: v_1, v_2, \dots, v_n)

T := tree consisting of v_1

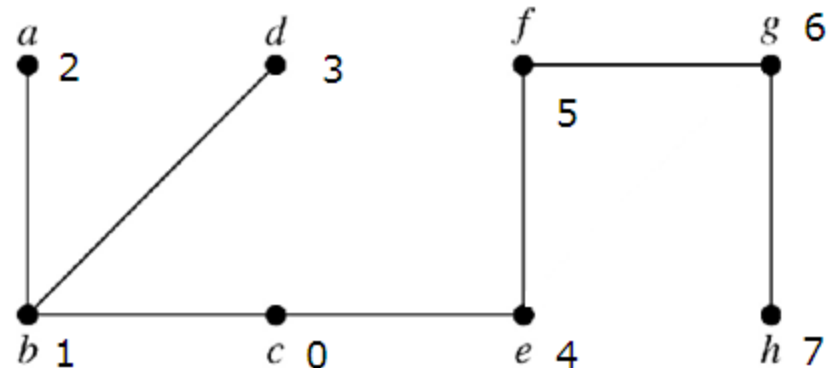
visit(v_1)

procedure visit(v : vertex of G)

for each vertex w adjacent to v and not yet in T

add vertex w and edge (v, w) to T

visit(w)



7. Start at c , breadth-first search for spanning tree.
Use alphabetically smaller letter first.

procedure BFS(G : graph, vertices: v_1, v_2, \dots, v_n)

T := tree consisting of v_1

L := empty list

put v_1 in the list L of unprocessed vertices

while L is not empty

 remove the first vertex, v , from L

for each neighbor w of v

if w not in L and not in T **then**

 add w to the end of list L

 add w and edge (v, w) to T

