CHAPTER 5

Trees and Binary Trees
Definition:

A tree is a finite set of one or more nodes such that:

1. There is a specially designated node called the root.

2. The remaining nodes are partitioned into $n \geq 0$ disjoint sets $T_1, \ldots, T_n$, where each of these sets is a tree.

3. We call $T_1, \ldots, T_n$ the subtrees of the root.
Example:

Root

Honey Bear

Brunhilde
  Gill
  Tansey
  Tweed
  Zoe

Terry

Brandy

Coyote
  Crocus
  Primrose

Nugget
  Nous
  Belle

Pedigree

Gill Tansey Tweed Zoe Crocus Primrose Nous Belle

Trees

NCUE CSIE Wireless Communications and Networking Laboratory
Trees

Terminology:

1. The **degree** of a node is the number of subtrees of the node.
2. The **degree of a tree** is the maximum of the degree of the nodes in the tree.
3. The node with degree 0 is a **leaf** or **terminal node**, and the other nodes are referred to as **nonterminals**.
4. A node that has subtrees is the **parent** of the roots of the subtrees.
5. The roots of these subtrees are the **children** of the node.
6. Children of the same parent are **siblings**.
7. The **ancestors** of a node are all the nodes along the path from the root to the node.
8. The **level** of a node is defined by letting the root be at level. If a node is at level $l$, then its children are at level $l+1$.
9. The **height** or **depth** of a tree is defined to be the maximum level of any node in the tree.
Trees

Terminology:

- node (13)
- root (A)
- degree of a node
- degree of a tree (3)
- leaf (terminal)
- nonterminal
- parent
- children
- sibling
- ancestor
- level of a node
- height of a tree (4)
Forests

**Definition:**

A forest is a set of $n \geq 0$ disjoint trees

**Example:**

- $\mathcal{F} = \{A, B, C, D\}
- \mathcal{F} = \{E, F\}
- \mathcal{F} = \{G, H, I\}$

*can be $\emptyset$*
**List Representation:**

| data | link 1 | link 2 | ... | link \(k\) |

- Store the node’s data.
- Pointing to the root of each subtree.
- The node(tree)’s degree is \(k\).
List Representation:

- (A (B (E (K, L), F), C (G), D (H (M), I, J)))
- The root comes first, followed by a list of sub-trees
Question

N nodes in a tree, the degree of tree is k.
1) Total number of the link field ?
2) The number of link field really used ?
3) The number of link field waste ?
4) What is the proportion of waste ?

Summary:

To reduce the proportion of the link field waste, then k = 2, the proportion of waste is approximately \( \frac{1}{2} \).
Definition:
A binary tree is a finite set of nodes that is either empty or consists of a root and two disjoint binary trees called the left subtree and the right subtree.

(1) Any tree can be transformed into binary tree by left child-right sibling representation.
(2) The left subtree and the right subtree are distinguished.

A

B

≠

A

B
Left Child - Right Sibling
Left Child - Right Sibling
typedef struct node *tree_pointer;
typedef struct node {
    int data;
    tree_pointer left_child, right_child;
};
Lemma:

The maximum number of nodes on level \( i \) of a binary tree is \( 2^{i-1} \), \( i \geq 1 \).

<table>
<thead>
<tr>
<th>Level</th>
<th>Max number of Nodes</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>( 2^0 )</td>
</tr>
<tr>
<td>2</td>
<td>( 2^1 )</td>
</tr>
<tr>
<td>3</td>
<td>( 2^2 )</td>
</tr>
<tr>
<td>4</td>
<td>( 2^3 )</td>
</tr>
</tbody>
</table>
Lemma:

The maximum number of nodes in a binary tree of depth $k$ is $2^k - 1$, $k \geq 1$.

Proof:

\[
\sum_{i=1}^{k} (\text{maximum number of nodes on level } i) = \\
\sum_{i=1}^{k} 2^{i-1} = \frac{2^0 (2^k - 1)}{2 - 1} = 2^k - 1
\]
Properties of Binary Trees

Lemma:

[Relation between number of leaf nodes and degree-2 nodes]:

For any nonempty binary tree, $T$, if $n_0$ is the number of leaf nodes and $n_2$ the number of nodes of degree 2, then $n_0 = n_2 + 1$

Example:

$n_0 = 3 = 2 + 1 = n_2 + 1$
Skewed Binary Trees

A
B
C
D
E

A
B

1
2
3
4
5
**Definition:**
A full binary tree of depth $k$ is a binary tree of depth $k$ having $2^k - 1$ nodes, where $k \geq 0$.

Full binary tree of depth 4
Definition:
A binary tree with \( n \) nodes and depth \( k \) is complete iff its nodes correspond to the nodes numbered from 1 to \( n \) in the full binary tree of depth \( k \).
Lemma:

If a complete binary tree with \( n \) nodes is represented sequentially, then for any node with index \( i, 1 \leq i \leq n \), we have:

1. parent(\( i \)) is at \( \left\lfloor \frac{i}{2} \right\rfloor \) if \( i \neq 1 \). If \( i = 1 \), \( i \) is at the root and has no parent.
2. left_child(\( i \)) is at \( 2i \) if \( 2i \leq n \). If \( 2i > n \), then \( i \) has no left child.
3. right_child(\( i \)) is at \( 2i + 1 \) if \( 2i + 1 \leq n \). If \( 2i + 1 > n \), then \( i \) has no right child.
Sequential Representation

A
B
C
D
E

[1] A
[2] B
[3] --
[4] C
[5] --
[6] --
[7] --
[8] D
[9] --
[16] E

A
B
C
D
E
F
G
H
I

[1] A
[2] B
[3] C
[4] --
[5] --
[6] --
[7] D
[8] --
[9] --
[16] E

NCUE CSIE Wireless Communications and Networking Laboratory
Binary Tree Traversals

Definition:

(1) Let L, V, and R stand for moving left, visiting the node, and moving right.
(2) There are six possible combinations of traversal
   LVR, LRV, VLR, VRL, RVL, RLV
(3) Adopt convention that we traverse left before right, only 3 traversals remain:
   LVR, LRV, VLR ←→ inorder, postorder, preorder
Theorem:

Give the only binary tree which Preorder/Inorder/Postorder can be determined.
void inorder(tree_pointer ptr)  
    /* inorder tree traversal */  
    {  
        if (ptr) {  
            inorder(ptr→left_child);  
            printf(“%d”, ptr → data);  
            inorder(ptr → right_child);  
        }  
    }
void preorder(tree_pointer ptr)
/* preorder tree traversal */
{
    if (ptr) {
        printf("%d", ptr → data);
        preorder(ptr → left_child);
        preorder(ptr → right_child);
    }
}
void postorder(tree_pointer ptr) {
    /* postorder tree traversal */
    if (ptr) {
        postorder(ptr → left_child);
        postdorder(ptr → right_child);
        printf("%d", ptr → data);
    }
}
procedure copy(original: tree_pointer): tree_pointer
{
    tree_pointer temp=nil;
    if (original≠nil)
    {
        temp→left_child=copy(original→left_child);
        temp→right_child=copy(original→right_child);
        temp→data=original→data;
    }
    return temp;
}
Equality of Binary Trees

```plaintext
procedure equal(first, second: tree_pointer): boolean
{
    equal=false
    if (first==nil) and (second==nil) then equal=true;
    else if (first≠nil) and (second≠nil) then
    {
        if (first →data == second →data) then
            if equal(first →left_child, second →left_child) then
                equal= equal(first →right_child, second →right_child);
    }
    return equal;
}
```
procedure count(T: tree_pointer):
{
    if (T≠nil) then
    {
        nL=count(T → left_child);
        nR=count(T → right_child);
        return (nL+nR+1);
    }
    else return 0
}
procedure height(T: tree_pointer):
{
    if (T≠nil) then
    {
        H_L=height(T → left_child);
        H_R=height(T → right_child);
        return max(H_L, H_R)+1;
    }
    else return 0
}

Height of Binary Trees
Arithmetic Expression Using BT

inorder traversal
A / B * C * D + E

infix expression
preorder traversal
+ * * / A B C D E

prefix expression
postorder traversal
A B / C * D * E +

postfix expression
A variable is an expression.

If $x$ and $y$ are expressions, then $\neg x$, $x \land y$, $x \lor y$ are expressions.

Parentheses can be used to alter the normal order of evaluation ($\neg > \land > \lor$).

Example: $x_1 \lor (x_2 \land \neg x_3)$

satisfiability problem: Is there an assignment to make an expression true?
Propositional Calculus Expression

\[(x_1 \land \neg x_2) \lor (\neg x_1 \land x_3) \lor \neg x_3\]

\( (t,t,t) \)
\( (t,t,f) \)
\( (t,f,t) \)
\( (t,f,f) \)
\( (f,t,t) \)
\( (f,t,f) \)
\( (f,f,t) \)
\( (f,f,f) \)

\(2^n \) possible combinations for \( n \) variables

postorder traversal (postfix evaluation)
node structure

<table>
<thead>
<tr>
<th>left_child</th>
<th>data</th>
<th>value</th>
<th>right_child</th>
</tr>
</thead>
</table>

typedef emun {not, and, or, true, false } logical;
typedef struct node *tree_pointer;
typedef struct node {
tree_pointer left_child;
logical data;
short int value;
tree_pointer right_child;
} ;
void post_order_eval(tree_pointer node)
{
    /* modified post order traversal to evaluate a propositional calculus tree */
    if (node) {
        post_order_eval(node->left_child);
        post_order_eval(node->right_child);
        switch(node->data) {
            case not: node->value = !node->right_child->value;
            break;
        }
    }
}
Post-order-eval function

case and:  node->value =
    node->right_child->value &&
    node->left_child->value;
break;

case or:    node->value =
    node->right_child->value ||
    node->left_child->value;
break;

    case true: node->value = TRUE;
             break;
    case false: node->value = FALSE;

}
Threaded Binary Trees

- Too many null pointers in current representation of binary trees
  - $n$: number of nodes
  - number of non-null links: $n-1$
  - total links: $2n$
  - null links: $2n-(n-1)=n+1$

- Replace these null pointers with some useful “threads”.
If `ptr->left_child` is null, replace it with a pointer to the node that would be visited before `ptr` in an inorder traversal.

If `ptr->right_child` is null, replace it with a pointer to the node that would be visited after `ptr` in an inorder traversal.
A Threaded Binary Tree

inorder traversal:
H, D, I, B, E, A, F, C, G
**Data Structures for Threaded BT**

<table>
<thead>
<tr>
<th>left_thread</th>
<th>left_child</th>
<th>data</th>
<th>right_child</th>
<th>right_thread</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRUE</td>
<td>•</td>
<td></td>
<td>•</td>
<td>FALSE</td>
</tr>
</tbody>
</table>

**TRUE: thread**

**FALSE: child**

typedef struct threaded_tree *threaded_pointer;

typedef struct threaded_tree {
    short int left_thread;
    threaded_pointer left_child;
    char data;
    threaded_pointer right_child;
    short int right_thread;
};
Memory Representation of A Threaded BT
threaded_pointer insucc(threaded_pointer tree)
{
    threaded_pointer temp;
    temp = tree->right_child;
    if (!tree->right_thread)
        while (!temp->left_thread)
            temp = temp->left_child;
    return temp;
}
void tinorder(threaded_pointer tree)
{
    /* traverse the threaded binary tree inorder */
    threaded_pointer temp = tree;
    for (;;) {
        temp = insucc(temp);
        if (temp==tree) break;
        printf("%3c", temp->data);
    }
}

O(n)
Heap

Definition:

(1) A *max tree* is a tree in which the key value in each node is no smaller than the key values in its children. A *max heap* is a complete binary tree that is also a max tree.

(2) A *min tree* is a tree in which the key value in each node is no larger than the key values in its children. A *min heap* is a complete binary tree that is also a min tree.
Heap

**Operation:**

1. Insert
2. Delete Max/Min
3. Create

**Application:**

Priority Queue
Heap

Example (Max Heap):

Property:
The root of max heap contains the largest value.
Heap

Example (Min Heap):

Property:
The root of min heap contains the smallest value.
Heap

Insert (Max Heap):
(1) Put X on the back of the last node (why?)
(2) Up to challenge the parent node, until fail or the parent node doesn’t exist.

initial location of new node
insert 5 into heap
insert 21 into heap
Delete (Max Heap):

1. Delete root.
2. Put the last node on root.
3. Adjust from root down.
Heap

Create (Max Heap):

(1) Top-Down: Continually perform “Insert”
(2) Bottom-Up: Heapify
Heapify

(1) First use data to establish the Complete Binary Tree.
(2) Adjust from last parent to root, until subtree all be Max-Heap

Example:
Please use “Bottom-Up” and the following data to establish Heap
26, 5, 77, 1, 61, 11, 59, 15, 48, 19,
## Summary

<table>
<thead>
<tr>
<th>Operation</th>
<th>Time Complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Insert</td>
<td>$O (\log n)$</td>
</tr>
<tr>
<td>Delete</td>
<td>$O (\log n)$</td>
</tr>
<tr>
<td>Search Max/Min</td>
<td>$O (1)$</td>
</tr>
<tr>
<td>Build</td>
<td>$O (n)$</td>
</tr>
</tbody>
</table>
Heapify

Time Complexity:

\[ \sum_{i=1}^{k-1} 2^{i-1} \times (k - i) \]

\[ \sum_{i=1}^{k-1} 2^{k-i-1} \times i \]

\[ \leq n \cdot \sum_{i=1}^{k-1} \frac{i}{2^i} < 2 \cdot n = O(n) \]

\[ 2^k - 1 = n \]
Binary Search Tree

**Definition:**

1. Every element has a unique key.
2. The keys in a nonempty left subtree (right subtree) are smaller (larger) than the key in the root of subtree.
3. The left and right subtrees are also binary search trees.

**Purpose:**

1. Search
2. Sorting
Examples of Binary Search Trees
Binary Search Tree

Build or Insert:

Insert 80

Insert 35
Delete:

1) First find the location of the X.
2) If X is the leaf, directly delete.
3) If X has a child up to replace X.
4) If X has a subtree, take the largest of right subtree or the smallest of left subtree to replace X, then go to (2).
Binary Search Tree

Example:

Before deleting 60

After deleting 60
Searching a Binary Search Tree

Procedure Search (BST tree_pointer T, x)
{
    if (T != nil) then
    {
        switch (compare x, T → data)
        {
            case '=' : return "found"
            case '<' : return Search (T → left_child, x)
            case '>' : return Search (T → right_child, x)
        }
        return "not found"
    }
    return "not found"
}
Time Complexity

(1) Worst case: \( O(n) \)
(2) Best case: \( O(\log n) \)

Example:

Full Binary Search Tree with height ‘\( h \)’:

\[
S = 2^0 \cdot 1 + 2^1 \cdot 2 + 2^2 \cdot 3 + \ldots + 2^{h-1} \cdot h \\
= -2^0 - 2^1 - 2^2 \ldots - 2^{h-1} + 2^h \cdot h \\
= 2^h \cdot h - 2^h + 1
\]

\[
\therefore \text{Ave.} = \frac{h \cdot 2^h - 2^h + 1}{2^h - 1}
\]
Question

Give a Binary Tree, please list: Preorder、Inorder、Postorder

Preorder: ABDECFG
Inorder: DBEAFCG
Postorder: DEBFGCA
Question

Give a Binary Tree, please list: Preorder, Inorder, Postorder

- Preorder: ACDEFG
- Inorder: DCAFGE
- Postorder: DCGFEA
Question

- Give an example of the Preorder/Postorder cannot be determined only binary tree

Preorder: ABC  Postorder: CBA
Question

- Based on the following data entry order to build Binary Search Tree
  26, 5, 33, 77, 19, 2, 13, 18

Answer:
If a binary tree with n Nodes, this binary tree
1) Maximum height?
2) Minimum height?

Key:

\[ K = \log_2 (n+1) \]
If a tree has a node of degree one, two nodes of degree two, three nodes of degree three....., $n$ nodes of degree $n$, how many leaf nodes are there in this tree?
Question

Write an algorithm, SwapTree(), that takes a binary tree and swaps the left and right children of every node.
Question

Binary Search Tree

(1) Delete 50
(2) Delete 5
(3) Delete 26
Continuous perform the following actions:

- Insert 80
- Insert 40
- Insert 100

Time Complexity: \(O(\log n)\)
Continuous perform “DELETE” twice.

Time Complexity: \( O(\log n) \)
Please use “Top-Down” and the following data to establish Heap.
26, 5, 77, 1, 61, 11, 59, 15, 48, 19,
Reference

- Ellis Horowitz, Sartaj Sahni, and Susan Anderson-Freed
- Ellis Horowitz, Sartaj Sahni, and Dinesh Mehta
  “Fundamentals of Data Structures in C++” Silicon Pr, 2006
- Richard F. Gilberg, Behrouz A. Forouzan,
- Fred Buckley, and Marty Lewinter
  “A Friendly Introduction to Graph Theory” Prentice Hall, 2002
- “資料結構-使用C語言” 蘇維雅譯，松崗，2004
- “資料結構-使用C語言” 蔡明志編著，全華，2004
- “資料結構(含精選試題)” 洪逸編著，鼎茂，2005