11 Trees

11.1 Introduction to Trees

Def. *1 A* ***tree*** $T = (V, E)$ *is*

a ***connected undirected graph*** *with* ***no simple circuits***.

acyclic connected *graph.*

forest *a* ***set*** *of* ***trees***.

A vertex of ***degree one*** *is called* ***leaf***.

Thm. *1 An undirected graph is a* ***tree***, *if and only if,*

there is a ***unique*** *simple* ***path*** *between* ***any*** *two of its vertices.*

proof *Assume T is a tree. Since,* ***connected***, $\forall u, v \in V, \exists$***path***$(u, v)$.

Assume $\exists$*another* ***path***(u, v). *Then* $\exists$***cycle***(u, v). $\therefore$ $\exists$***unique path***(u, v).

Assume $\exists$***unique path***(u, v). *T is* ***connected*** *and no* ***simple circuit***.

$\therefore$ ***unique path*** *between every two vertices is a* ***tree***.*(another* ***Def****. of tree)*

Def. *2 A* ***rooted tree*** $T_r = (V, E, r)$ *is a tree where*

one vertex has designated as the ***root****(*$r \in V$*)*

and every edge is ***directed away*** *from the root.*

parent, ***children***, ***siblings*** ***ancestor***, ***decendents***

leaves, ***internal vertices*** ***subtree***

Recursive *definition of rooted tree(see 5.3* ***Def. 3****) in* ***digraph***

Def. *3 A rooted tree is called* ***m-ary tree***,

if ***every (internal)*** *vertex has* ***no more than*** *m children.*

The tree is called ***full m-ary*** *tree,*

if ***every internal****(non-leaf) vertex has* ***exactly*** *n children.*

A 2-ary tree is call ***binary*** *tree.*

An ***orderd rooted tree***

children are ordered from left to right

left/right *child*

left/right *subtree*

Tree as Models

Computer file system

Hierarchical structure

Family tree

Properties of Trees

Thm. *2 A tree with n* ***vertices*** *has n–1* ***edges***. $e = n - 1$.

minimal connections of vertices

Thm. *3 A* ***full*** *m-ary tree with i* ***internal vertices*** *has* $n = mi+1$ ***vertices***.

and $l = (m-1)i + 1$ ***leaves***.

proof *Every vertices except* ***root*** *is a child of internal vertices.*

$\therefore n = mi + 1$.

Since $n = i + l$*(leaves),* $i + l = mi + 1$. $\therefore l = (m-1)i + 1$ *leaves.*

Thm. *4 A* ***full*** *m-ary tree with e, i, n, l.*

$e = n - 1$ $n = i + l$ $n = mi + 1$.

(1) n vertices has $i = (n-1)/m$ *internal vertices and* $l = [(m-1)n+1]/m$ *leaves.*

(2) i intertinal vertices has $n = mi+1$ *vertices and* $l = (m-1)i + 1$ *leaves.*

(3) l leaves has $n = (ml-1)/(m-1)$ *vertices and* $i = (l-1)/(m-1)$ *interior vertices.*

*The **level** of a **node** is the **length** of the **path** from **root** to the node.*
*The **height** of a **tree** is **maximum** node **level**.*
*The m-ary tree with height h is called **balanced**,*
*if every **leaves** are **level** h or h–1.*

***Thm**. 5 There are **at most** m^h **leaves** in m-ary tree of **height** h.*
***proof** Consider a **full**, **balanced** m-ary tree*
*and **every leaves** are of **level** h.*
*It has m^h **leaves**.*

***Col**. 1 If an m-ary tree with l leaves has hight h, then $h \geq \lceil \log_m l \rceil$.*
*If the tree is **full** and **balanced** $h = \lceil \log_m l \rceil$.*

Thm. 6 *Every* ***tree*** *has following properties:*

1. Any ***connected subgraph*** *is a* ***tree****.*

2*. There is a* ***unique*** *(****simple****)* ***path*** *between* ***every pair of vetices****.*

3. ***Adding*** *an* ***edge*** *between two vertices create a (****simple****)* ***cycle****.*

4. ***Removing*** *any* ***edge disconnects*** *the tree.*

5. If tree has at ***least two vertices****, then it has* ***at least two leaves****.*

6*. The number of* ***vertices*** *is* ***one larger*** *than that of* ***edges****.*

proof *1. Any subgraph of acyclic graph subgraph is also* ***acyclic****.*

2. There is ***at least one*** *(simple) path, since connected and acyclic.*

Assume ***two different*** *simple paths from u to v.*

Assume x be the first vertex where the path ***diverge****,*

y be the next vertex they ***share****.*

There is a ***cycle*** *from x to y and then y to x.*

3. Additional edge $\{u, v\} \cup$ *(simple)* ***path*** *from u to v = (simple)* ***cycle***

*4. Remove {u, v}. **Unique** simple path was (u, v). ∴Not connected*

*5. Let $(v_1, \ldots, v_m)$ be the **longest** simple path in the tree. Then $m \geq 2$.*

*$2 < \forall i \leq m, \{v_1, v_i\} \notin E$, since $(v_1, \ldots, v_i, v_1)$ is a **cycle**.*

*$\{u, v_1\} \notin E$ where u is not in the path, since $(v_1, \ldots, v_m)$ is the **longest**.*

*$\therefore$ v_1 is a **leaf**. By **symmetric** argumant v_m is a **second** leaf.*

6. Induction on number of vertices.

n=1, no edge. 0 + 1 = 1. O.K.

*Consider (n+1)-vertex tree T and let v be a **leaf** of T.*

*Deleting v and its incident edge gives a smaller **tree**.*

$(|E| - 1) = (|V| - 1) + 1$

*Adding v and its incident edge gives a larger **tree**. $\therefore |E| = |V| + 1$.*

11.2 Application of Trees

Binary Search Tree

procedure search&insert(T = (V, E, r): binary Tree, x: item)
if $r \neq null \rightarrow v := r;$
  do $v \neq null \wedge label(v) \neq x \rightarrow$
    if $x < label(v) \rightarrow$
      if $left\ child(v) \neq null \rightarrow v := left\ child(v)$
       | $left\ child(v) = null \rightarrow$ add new vertex v(label(x)) as a left child
      fi
     | $x > label(v) \rightarrow$
      if $right\ child(v) \neq null \rightarrow v := right\ child(v)$
       | $right\ child(v) = null \rightarrow$ add new vertex v(label(x))as a right child
  od fi fi
 | $r = null \rightarrow$ add new vertex v(label(x)) as root of the tree
fi

label(v) = x.

```
procedure search&insert(T = (V, E, r): binary Tree, x: item)
    v := r;
    if v ≠ null →
        if x < label(v) → return search&insert(leftSubtree(T), x)
         | x > label(v) → return search&insert(rightSubtree(T), x)
         | x = label(v) → return "arleady present"
        fi
     | v = null →
        r := add a new vertex with label x;
        return "Not present and inserted"
    fi
```

Decision Trees

*One **counterfeit**(함량불량) coin in n coins*

balance

Three groups of $\lceil n/3 \rceil$ coins

***if** first group < second group → **counterfeit** in the first group*

*| first group = second group → **counterfeit** in the third group*

*| first group > second group → **counterfeit** in the second group*

fi

$\lceil \log_3 n \rceil$ *comparisons.*

***Ex** 3. **Fig**. 3 One **counterfeit** coin and 7 **normal** coins.(p730)*

$\lceil \log_3 8 \rceil = 2$ *comparisons!*

The complexity of sorting algorithms

n elements

n! permutations

binary comparisions

n! leaves in the binary decision tree

Height of the decision tree is log n!

$O(n \log_2 n)$ is ***optimal***

The ***lower bound*** *for sorting based on binary search is $\Omega(n \log_2 n)$*

The ***average*** *bound for sorting based on binary search is $\Theta(n \log_2 n)$*

Prefix code

26 leters *5 bits*($2^4 < 26 \leq 2^5$)

But more ***frequent*** *letters have* ***shorter*** *sequence.*

e *00*

t *01*

w *0001*

0001 *w or et*

e is a prefix of w

A set of sequences(codes) is a ***prefix code****, if no sequence in the set is the* ***prefix*** *of other sequences in the set.*

Full binary tree $\leftrightarrow$ *prefix code*

Algorithm** 2 **Huffman coding

***procedure** Huffman(V: set of symbols, w: $V \rightarrow \mathbf{R}^+$ is a weigh of a symbol)*

*F := **for** $a \in V$ **do** make a **tree** with symbol a and weight $w(a)$ **od***

***do** F is **not** a tree $\rightarrow$*

*Find **two minimun** weighted trees T_1 and T_2;*

*Make a **new tree** T with weight $w(T) = w(T_1) + w(T_2)$*

*and T_1 and T_2 as **subtrees**.*

od

internal vertices	*weight of the subtree*
leaf	*symbol*

*See **Fig**. 6 Huffman coding in p 734.*

*Is the Huffman coding **optimal**?*

11.3 Tree Traversal

Universal Address System in a Rooted Ordered Tree

Label root as 0.

If Subtree T is labeled as x and n children

then n children of x are labeled as x.1, x.2, …, x.n.

We use x instead of 0.x(omit 0.).

***Ex** 1 rooted tree (**Fig**.1 p741)*

0 < 1 < 1.1 < 1.2 < 1.3 < 2 < 3 < 3.1 < 3.1.1 < 3.1.2 < 3.1.2.1 < 3.1.2.2 < 3.1.2.3 < 3.1.2.4 < 3.1.3 < 3.2 < 4 < 4.1 < 5 < 5.1 < 5.1.1 < 5.2 < 5.3

Prefix code

Lexicographic ordering

$x_1.x_2.\ldots.x_n < y_1.y_2.\ldots.y_m$, *if* $\exists i,\ 1 \le i \le n,\ 1 \le \forall j < i,\ x_j = y_j,\ x_i < y_i$ *or*

$n < m,\ 1 \le \forall j \le n,\ x_j = y_j.$

procedure *preorder(T with root r)*

visit r; preorder(child$_1$(r)); preorder(child$_2$(r)); …; preorder(child$_n$(r))

end *preorder;*

0 < 1 < 1.1 < 1.2 < 1.3 < 2 < 3 < 3.1 < 3.1.1 < 3.1.2 < 3.1.2.1 < 3.1.2.2 < 3.1.2.3 < 3.1.2.4 < 3.1.3 < 3.2 < 4 < 4.1 < 5 < 5.1 < 5.1.1 < 5.2 < 5.3

Same as ***lexicographic*** *order*

procedure *inorder(T with root r)*

inorder(child$_1$(r)) visit r; inorder(child$_2$(r)); … inorder(child$_n$(r))

end *inorder;*

1.1 < 1 < 1.2 < 1.3 < 0 < 2 < 3.1.1 < 3.1 < 3.1.2.1 < 3.1.2 < 3.1.2.2 < 3.1.2.3 < 3.1.2.4 < 3.1.3 < 3 < 3.2 < 4.1 < 4 < 5.1 < 5.1.1 < 5 < 5.2 < 5.3

procedure *postorder(T with root r)*

postorder(child$_1$(r)); posteorder(child$_2$(r)); … postorder(child$_n$(r)); visit r

end *postorder*

1.1 < 1.2 < 1.3 < 1 < 2 < 3.1.1 < 3.1.2.1 < 3.1.2.2 < 3.1.2.3 < 3.1.2.4 < 3.1.2 < 3.1.3 < 3.1 < 3.2 < 3 < 4.1 < 4 < 5.1.1 < 5.1 < 5.2 < 5.3 < 5 < 0

Infix Notation $x \oplus y$

binary operation *binary expression tree*

inorder *traversal of the expression tree*

Ambiguous ***parentheses*** *()*

precedence *between operators* $x \oplus y \otimes z$

$prec(\uparrow) > prec(\times) = prec(/) > prec(+) = prec(-)$

$u \uparrow x - y \times z = (u \uparrow x) - (y \times z))$

associativity of operands $x \oplus y \oplus z$

left *associative* $\times$, $+$ $x \times y \times z = (x \times y) \times z$

right *associative* $\uparrow$, $/$, $-$ $x \uparrow y \uparrow z = x \uparrow (y \uparrow z)$

Prefix(Polish) Notation $\oplus_n op_1 op_2 \ldots op_n$

n-ary opertaion *n-ary expression tree*

preorder *traversal of the expression tree*

Postfix(reversed Polish) Notation $op_1 op_2 \ldots op_n \oplus_n$

operational ***stack***

*Ambiguous but simple **context-free** grammar for binary infix expressions*

$$E \to E + E \mid E - E \mid E \times E \mid E / E \mid E \uparrow E \mid a \mid \ldots \mid z \mid (E)$$

*Unambiguous **context-free** grammar (without **parentheses**)*

$$E \to E + T \mid T - E \mid T \times F \mid F / T \mid X \uparrow F \mid a \mid \ldots \mid z \mid (E)$$
$$T \to T \times F \mid F / T \mid X \uparrow F \mid a \mid \ldots \mid z \mid (E)$$
$$F \to X \uparrow F \mid a \mid \ldots \mid z \mid (E)$$
$$X \to a \mid \ldots \mid z \mid (E)$$

*Consider **precedence** between operators: for example* $u \uparrow x - y \times z$.

$\underline{E} \Rightarrow \underline{T} - E \Rightarrow \underline{X} \uparrow F - E \Rightarrow u \uparrow \underline{F} - E \Rightarrow u \uparrow x - \underline{E} \Rightarrow u \uparrow x - \underline{T} \times F \Rightarrow u \uparrow x - y \times \underline{F} \Rightarrow u \uparrow x - y \times z.$

*Consider **associativity** of operators: for example* $x \uparrow y \uparrow z$.

$\underline{E} \Rightarrow \underline{X} \uparrow F \Rightarrow x \uparrow \underline{F} \Rightarrow x \uparrow \underline{X} \uparrow F \Rightarrow x \uparrow y \uparrow \underline{F} \Rightarrow x \uparrow y \uparrow z.$

*Compare **grammar tree** and **expression tree***

11.4 Spanning Trees

Def. 1 Let G be a ugraph. A **spanning tree** of G is a subgraph of G that is **tree** containing **every vertex** of G.

Thm. 1 Every **connected graph** has **spanning tree**.

proof Let T be a **connected spanning** subgraph of G with the **smallest** number of **edges**.

Suppose T has a cycle $(v_0, v_1, \ldots, v_n, v_0)$.

Suppose we remove the edge $\{v_n, v_0\}$.

If arbitrary vertices x and y has a **path not** containing the edge (v_n, v_0), x and y has a path containing **that path**.

If arbitrary vertices x and y has a path **containing** the edge (v_n, v_0), x and y has a path containing the **path** $(v_0, v_1, \ldots, v_n)$.

This is a **contradiction** that T has the **smallest** number of edges and **connected**.

$\therefore$ T is acyclic. $\therefore$ T is a tree.

Alg**. 1 **Depth-First Search

***procedure** DFS(G = (V, E): connected graph)*

T := Tree with vertex v ∈ V only;

visit(v);

***end** DFS;*

***procedure** visit(v ∈ V)*

for** (v, w) ∈ E ∧ w ∉ T **do** add w and (v, w) to T; visit(w) **od

***end** visit*

Alg**. 2 **Breadth-First Search

***procedure** BFS(G = (V, E): connected graph)*

T := Tree with vertex v ∈ V only; L := list with vertex v∈ V;

***do** L is **not** empty →*

remove the first vertex, v, from L;

*for (v, w) ∈ E **where** w ∉ T ∧ w ∉ L **do***

*add w to the end of L; add w and (v, w) to T **od od***

***end** BFS;*

11.5 Minimum Spanning Trees

***Definition** 1 A **minimum panning tree** in a connected weighted graph is a spanning tree that has the **smallest sum of weight** of its edges.*

Algorithm** 1 **Prim's Algorithm

***procedure** Prim(G = (V, E): connected weighted graph)*

T := a minimum weighted edge;

for** i := 1 to |V| – 2 **do

*e := a **minimum** weighted edge **incident** to a vertex in T*

*and **not** forming a simple circuit in T, if added to T;*

T := T with e added

od

***end** Prim*

Algorithm 2 Kruskal's Algorithm

procedure *Kruskal(G = (V, E): connected weighted graph)*

T := empty graph;

for *i := 1 to |V| – 1* ***do***

e := any edge in G with ***smallest*** *weight*

that does ***not*** *form a simple circuit in T, if added to T;*

T := T with e added

od