Trees (II)

Hwansoo Han
Abstract Data Type - Tree

- Operations on Tree $T$
  - PARENT ($n, T$) – parent of $n$
  - LEFTMOST_CHILD ($n, T$) – leftmost child of $n$
  - RIGHT_SIBLING ($n, T$) – immediately to the right of $n$
  - LABEL ($n, T$) – label of $n$
  - CREATE$i$ ($v, T_1, T_2, ..., T_i$) – make a node $r$ with label $v$ and gives it $i$ children, which are the roots of $T_1, T_2, ..., T_i$
  - ROOT ($T$) – root of $T$
  - MAKENULL ($T$) – make $T$ be the null tree
Example – Preorder Traversal

```plaintext
procedure PREORDER ( n: node );
    { list the labels of the descendants of n in preorder }
var
c: node;
begin
    print(LABEL(n, T));
c := LEFTMOST_CHILD(n, T);
while c <> Λ do
begin
    PREORDER(c);
c := RIGHT_SIBLING(c, T)
end
end;    { PREORDER }
```
Implementation of Trees

- Array representation of trees

```haskell
type
    node = integer;
TREE = array [1..maxnodes] of node;
```
Array Representation of Trees - Example

- Array elements
  - Specify parent node
Array Representation of Trees - Example

- RIGHT_SIBLING \((n, T)\)

```plaintext
function RIGHT_SIBLING (n: node; T: TREE): node;
{ return the right sibling of node \(n\) in tree \(T\) }

var
  i, parent: node;

begin
  parent := T[n];
  for i := n + 1 to maxnodes do
    { search for node after \(n\) with same parent }
    if T[i] = parent then
      return (i);
    return (0) { null node will be returned if no right sibling is ever found }
end; { RIGHT_SIBLING }
```
Implementation of Trees

- Representation of Trees by Lists of Children
  - For each node, form a list of its children

```plaintext
type
  node = integer;
  LIST = { appropriate definition for list of nodes };
  position = { appropriate definition for positions in lists };
  TREE = record
    header: array [1..maxnodes] of LIST;
    labels: array [1..maxnodes] of labeltype;
    root: node
  end;
```
Trees by Lists of Children - Example
function LEFTMOST_CHILD ( \( n: \) node; \( T: \) TREE): node;
{ returns the leftmost child of node \( n \) of tree \( T \) }

var

\( L: \) LIST; \{ shorthand for the list of \( n \)'s children \}

begin

\( L := T.\text{header}[n]; \)

if EMPTY\( (L) \) then \{ \( n \) is a leaf \}

return \( (0) \)

else

return \( (\text{RETRIEVE}(\text{FIRST}(L), L)) \)

end; \{ LEFTMOST_CHILD \}
Trees by Lists of Children – Another Impl.
Binary Trees

- Definition
  - Every node has either no children, a *left child*, a *right child*, or both a left and a right child
  - Empty tree is also a binary tree
Binary Trees - Example

Two Binary Trees

A Ordinary Tree
Representation of Binary Trees

- Use an array for nodes, $1, 2, ..., n$

```
var
cellspace : array [1..maxnodes] of record
  leftchild: integer;
  rightchild: integer;
end;
```
## Representation of Binary Trees

<table>
<thead>
<tr>
<th>left child</th>
<th>right child</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>left child</th>
<th>right child</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
</tr>
</tbody>
</table>

Two Binary Trees
Binary Trees Example – Huffman Codes

- A message made from five characters
  - a, b, c, d, e
  - which appear with probabilities, .12, .4, .15, .08, .25

- Encode each character into a sequence of 0’s and 1’s
  - No code for a character is not the prefix of the code for any other character – *prefix property*
Two possible codes both of which satisfy prefix property

- bcd – Code1: 001010011
  – Code2: 1101001
Codes with Prefix Property - Example

- Binary trees representing codes with prefix property

![Binary trees representing codes with prefix property](image-url)
Huffman Codes

- A set of characters and their probabilities are given,
  - Find a code with the prefix property such that the average length of a code for a character is a minimum

- Average length of a character for previous codes
  - Code1: 3 as all characters are coded with 3 bits
  - Code2: 3 for $a, d$ (total 20%), 2 for $b, c, e$ (total 80%)
    - average is $2.2 = 3 \times 20\% + 2 \times 80\%$
Construction of a Huffman Tree

(a) Initial

(b) Merge a and d

(c) Merge a, d with c

(d) Merge a, c, d with c

(e) Final
Huffman Codes - Example

- Find an optimal Huffman code for
  - Alphabet = \{a, b, c, d, e, f\}
  - Probabilities = 0.07, 0.09, 0.12, 0.22, 0.23, 0.27