Irregular Data Structures

Linked lists, trees, and graphs
Linked list

• A 1-D sequence data structure
• Not an array
• Are created Dynamically, unlike Arrays which are created statically
• Each data element is linked to the next
• Link: memory address pointing to a data element
• Link list node: data element + link
Linked list (Adding a node)

• Assume, we want to maintain class grades in descending order

• To add a new grade with value of 89
  – Start at first node, also called the head node (using known red arrow link called the head pointer)
  – Check if amount (100) smaller than 89?
  – No, use node link to go to next node
  – Is amount (94) smaller than 89?
  – No, use node link to go to next node
  – Is amount (85) smaller than 89?
Linked list

• Add a new transaction in the amount of $8.12
  – Yes, insert new node
    • Make new node
    • Set new node value to 89
    • Set new node link to next node
    • Set previous node link to new node
Linked list (Deleting a node)

• Delete grades 94
  – Move to node storing the grade 94, starting from the head pointer and traversing the link of the nodes till you find the node with grade 94
  – Travelling through the blue link will make you reach 94
Linked list

• Delete the node with grade 94 by:
  – Setting link of previous node of 94 to next node of 94
  – Deleting current node after modifying the link
Linked list (Deleting the head node)

• Delete the node with grade value 99
  – Special case
  – Set red link equal to node pointed by link of first node (node pointed by the blue link)
  – Delete first node
Binary tree

• Definition
  – A hierarchical data structure
  – A (parent) node links to 0, 1, or 2 (children) nodes
  – The starting node is called root; the root is not the child of any node
  – Nodes with 0 children are called leaf nodes
  – Non-leaf nodes are called internal nodes
Application of Binary Trees: Arithmetic expression

- Operators stored at internal nodes
- Operands stored at leafs
- Arithmetic expression can be recovered by traversing tree
- Traversal: visiting all nodes
- Traversal rules
  - Start at root
  - For every node
    - go left until dead end
    - then go right until dead end
    - then go back up
- Printout rules
  - Write “(“ before going left
  - Write current node symbol before going right
  - Write “)” after having gone right
Application of Binary Trees: Arithmetic expression

- Operators stored at internal nodes
- Operands stored at leafs
- Arithmetic expression can be recovered by traversing tree
- Traversal: visiting all nodes
- Traversal rules
  - Start at root
  - For every node
    - go left until dead end
    - then go right until dead end
    - then go back up
- Printout rules
  - Write “(“ before going left
  - Write current node symbol before going right
  - Write “)” after having gone right
Application of Binary Trees: Arithmetic expression

- Operators stored at internal nodes
- Operands stored at leafs
- Arithmetic expression can be recovered by traversing tree
- Traversal: visiting all nodes
- Traversal rules
  - Start at root
  - For every node
    - go left until dead end
    - then go right until dead end
    - then go back up
- Printout rules
  - Write “(“ before going left
  - Write current node symbol before going right
  - Write “)’’ after having gone right
Application of Binary Trees: Arithmetic expression

- Operators stored at internal nodes
- Operands stored at leafs
- Arithmetic expression can be recovered by traversing tree
- Traversal: visiting all nodes
- Traversal rules
  - Start at root
  - For every node
    - go left until dead end
    - then go right until dead end
    - then go back up
- Printout rules
  - Write “(“ before going left
  - Write current node symbol before going right
  - Write “)” after having gone right

```
(((9
```

```
+ 9 5
*
/
```

```
- 6 1
```

```
4 2
```

```
- 1
```

```
9
```
Application of Binary Trees: Arithmetic expression

- Operators stored at internal nodes
- Operands stored at leafs
- Arithmetic expression can be recovered by traversing tree
- Traversal: visiting all nodes
- Traversal rules
  - Start at root
  - For every node
    - go left until dead end
    - then go right until dead end
    - then go back up
- Printout rules
  - Write “(“ before going left
  - Write current node symbol before going right
  - Write “)” after having gone right

(((9+5)*4)-2/6)
Application of Binary Trees: Arithmetic expression

- Operators stored at internal nodes
- Operands stored at leaves
- Arithmetic expression can be recovered by traversing tree
- Traversal: visiting all nodes
- Traversal rules
  - Start at root
  - For every node
    - go left until dead end
    - then go right until dead end
    - then go back up
- Printout rules
  - Write “(“ before going left
  - Write current node symbol before going right
  - Write “)’” after having gone right

(((9+5)−2)∗6)-1
Application of Binary Trees: Arithmetic expression

- Operators stored at internal nodes
- Operands stored at leafs
- Arithmetic expression can be recovered by traversing tree
- Traversal: visiting all nodes
- Traversal rules
  - Start at root
  - For every node
    - go left until dead end
    - then go right until dead end
    - then go back up
- Printout rules
  - Write “(“ before going left
  - Write current node symbol before going right
  - Write “)” after having gone right

\[((9+5)\)\]
Application of Binary Trees: Arithmetic expression

- Operators stored at internal nodes
- Operands stored at leafs
- Arithmetic expression can be recovered by traversing tree
- Traversal: visiting all nodes
- Traversal rules
  - Start at root
  - For every node
    - go left until dead end
    - then go right until dead end
    - then go back up
- Printout rules
  - Write “(“ before going left
  - Write current node symbol before going right
  - Write “)” after having gone right

(((9+5)*
Application of Binary Trees: Arithmetic expression

- Operators stored at internal nodes
- Operands stored at leafs
- Arithmetic expression can be recovered by traversing tree
- Traversal: visiting all nodes
- Traversal rules
  - Start at root
  - For every node
    - go left until dead end
    - then go right until dead end
    - then go back up
- Printout rules
  - Write "(" before going left
  - Write current node symbol before going right
  - Write ")" after having gone right

(((9+5)*6)-(1)
Application of Binary Trees: Arithmetic expression

- Operators stored at internal nodes
- Operands stored at leafs
- Arithmetic expression can be recovered by traversing tree
- Traversal: visiting all nodes
- Traversal rules
  - Start at root
  - For every node
    - go left until dead end
    - then go right until dead end
    - then go back up
- Printout rules
  - Write “(“ before going left
  - Write current node symbol before going right
  - Write “)“ after having gone right

(((9+5)*(4-2)-6+1)
Application of Binary Trees: Arithmetic expression

- Operators stored at internal nodes
- Operands stored at leafs
- Arithmetic expression can be recovered by traversing tree
- Traversal: visiting all nodes
- Traversal rules
  - Start at root
  - For every node
    - go left until dead end
    - then go right until dead end
    - then go back up
- Printout rules
  - Write “(“ before going left
  - Write current node symbol before going right
  - Write “)” after having gone right

(((9+5)*(4-6)-1)}
Application of Binary Trees: Arithmetic expression

- Operators stored at internal nodes
- Operands stored at leafs
- Arithmetic expression can be recovered by traversing tree
- Traversal: visiting all nodes
- Traversal rules
  - Start at root
  - For every node
    - go left until dead end
    - then go right until dead end
    - then go back up
- Printout rules
  - Write “(" before going left
  - Write current node symbol before going right
  - Write ")" after having gone right

\(((9+5)*(4-2)\)
Application of Binary Trees: Arithmetic expression

- Operators stored at internal nodes
- Operands stored at leafs
- Arithmetic expression can be recovered by traversing tree
- Traversal: visiting all nodes
- Traversal rules
  - Start at root
  - For every node
    - go left until dead end
    - then go right until dead end
    - then go back up
- Printout rules
  - Write “(“ before going left
  - Write current node symbol before going right
  - Write “)” after having gone right

((((9+5)*(4-2))))
Application of Binary Trees: Arithmetic expression

- Operators stored at internal nodes
- Operands stored at leafs
- Arithmetic expression can be recovered by traversing tree
- Traversal: visiting all nodes
- Traversal rules
  - Start at root
  - For every node
    - go left until dead end
    - then go right until dead end
    - then go back up
- Printout rules
  - Write "(" before going left
  - Write current node symbol before going right
  - Write ")" after having gone right

(((9+5)*(4-2)))
Application of Binary Trees: Arithmetic expression

- Operators stored at internal nodes
- Operands stored at leafs
- Arithmetic expression can be recovered by traversing tree
- Traversal: visiting all nodes
- Traversal rules
  - Start at root
  - For every node
    - go left until dead end
    - then go right until dead end
    - then go back up
- Printout rules
  - Write "(" before going left
  - Write current node symbol before going right
  - Write ")" after having gone right

```
(((9+5)*(4-2))/1
```
Application of Binary Trees: Arithmetic expression

- Operators stored at internal nodes
- Operands stored at leafs
- Arithmetic expression can be recovered by traversing tree
- Traversal: visiting all nodes
- Traversal rules
  - Start at root
  - For every node
    - go left until dead end
    - then go right until dead end
    - then go back up
- Printout rules
  - Write “(“ before going left
  - Write current node symbol before going right
  - Write “)” after having gone right

(((9+5)*(4-2)))/(24+9)
Application of Binary Trees: Arithmetic expression

- Operators stored at internal nodes
- Operands stored at leafs
- Arithmetic expression can be recovered by traversing tree
- Traversal: visiting all nodes
- Traversal rules
  - Start at root
  - For every node
    - go left until dead end
    - then go right until dead end
    - then go back up
- Printout rules
  - Write “(“ before going left
  - Write current node symbol before going right
  - Write “)“ after having gone right

\[
(((9+5)*(4-2)))/(6)
\]
Application of Binary Trees: Arithmetic expression

- Operators stored at internal nodes
- Operands stored at leafs
- Arithmetic expression can be recovered by traversing tree
- Traversal: visiting all nodes
- Traversal rules
  - Start at root
  - For every node
    - go left until dead end
    - then go right until dead end
    - then go back up
- Printout rules
  - Write “(“ before going left
  - Write current node symbol before going right
  - Write “)” after having gone right

(((9+5)*(4-2))/(6-1)
Application of Binary Trees: Arithmetic expression

• Operators stored at internal nodes
• Operands stored at leafs
• Arithmetic expression can be recovered by traversing tree
• Traversal: visiting all nodes
• Traversal rules
  – Start at root
  – For every node
    • go left until dead end
    • then go right until dead end
    • then go back up
• Printout rules
  – Write "(" before going left
  – Write current node symbol before going right
  – Write ")" after having gone right

(((9+5)*(4-2))/(6-1)
Application of Binary Trees: Arithmetic expression

- Operators stored at internal nodes
- Operands stored at leafs
- Arithmetic expression can be recovered by traversing tree
- Traversal: visiting all nodes
- Traversal rules
  - Start at root
  - For every node
    - go left until dead end
    - then go right until dead end
    - then go back up
- Printout rules
  - Write “(“ before going left
  - Write current node symbol before going right
  - Write “)“ after having gone right

(((9+5)*(4-2))/(6-1))
Application of Binary Trees: Arithmetic expression

- Operators stored at internal nodes
- Operands stored at leaves
- Arithmetic expression can be recovered by traversing tree
- Traversal: visiting all nodes
- Traversal rules
  - Start at root
  - For every node
    - go left until dead end
    - then go right until dead end
    - then go back up
- Printout rules
  - Write “(“ before going left
  - Write current node symbol before going right
  - Write “)” after having gone right

$$(((9+5)*(4-2))/(6-1))$$
Graphs

- Graphs
  - Nodes (also called vertices) connected by links, called edges
  - Nodes have a variable number of incident edges
  - Great flexibility
- Example: Facebook ‘like’
Graph data structure encodings – List of Edges

- "List of edges"
  - Array of nodes & array of edges
  - Edges pair of node indices
  - Origin node first
  - Destination node second

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Marry</td>
<td>Jane</td>
<td>Jim</td>
<td>Frank</td>
<td>John</td>
<td>Lucy</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>0,3</td>
<td>3,0</td>
<td>1,0</td>
<td>4,3</td>
<td>3,4</td>
<td>4,1</td>
<td>4,2</td>
<td>4,5</td>
<td>3,1</td>
<td>5,2</td>
</tr>
</tbody>
</table>
Graph data structure encodings – Adjacency List

• “Adjacency lists”
  – One array for each node
  – Array stores adjacent nodes
  – Finding an edge only requires traversing the starting node’s adjacency list

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Marry</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Jane</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Jim</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Frank</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>John</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Lucy</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Adjacency list for node 0

Adjacency list for node 1

Adjacency list for node 2

Adjacency list for node 3

Adjacency list for node 4

Adjacency list for node 5
Graph data structure encodings – Adjacency Matrix

• “Adjacency matrix”
  – A 2-D matrix
  – Row corresponds to start node
  – Column corresponds to end node
  – 0 if no edge, 1 if edge

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Marry</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Jane</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Jim</td>
<td>2</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Frank</td>
<td>3</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>John</td>
<td>4</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>Lucy</td>
<td>5</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>