Computer Science for Engineers

Lecture 8

Data structures – part 2

Prof. Dr. Dr.-Ing. Jivka Ovtcharova

Dipl. Wi.-Ing. Dan Gutu

18th of December 2009
• General dynamic system: mechanical, electrical, hydraulic, thermal, chemical systems, among others

• Example of a dynamic system: traveling vehicle, micro-electronic switch, satellite positioning system

• Classic modeling approach
  - Creation of an ideal model of a real system
  - Transformation of the ideal model into mathematical equations, esp. „block diagram“.

• Disadvantages to the classic modeling approach
  - Elaborate process
  - With new changes to the ideal model, new equations must then be derived
  - Discipline specific approach (mechanic, electric, hydraulic …)
The Bond Graph Approach

- Founded by H. Paynter, MIT, MA, 1959
- Recursive top-down disassembly into sub systems that exchange energy with one another
- Interdisciplinary representation of the dynamic behavior of physical systems
- System models are described with a uniform notation for all types of physical systems
- A powerful tool for modeling technical systems, esp. when more than one discipline is involved.
- Contains the information about the physical structure: the structure of a bond graph results from the topologic structure of the schematic representation of the system.
- With changes to the ideal model, only the concerned model parts must be updated: very good for model driven design and "What-If?“ situations
- The Bond graph-approach is an object oriented approach
Where can bond graphs be used?

- Thermodynamic
- Mechanic
- Hydraulic
- Electric
- Magnetism and chemical
Usage Areas of Bond Graphs

4. Data Structures
4.3 Use of Graphs

- Electric
- Mechanic Translation
- Mechanic Rotation
- Hydraulic/Pneumatic
- Thermodynamic
- Chemical engineering

Chemical engineering

\[ I_s = I_{11} \frac{\Delta T}{T} + I_{12} \frac{\Delta \mathcal{E}}{T} \]
A tree represents a special kind of graph.

- a tree is a graph in which additional conditions must be given (similar to: every rectangle is a polygon).
- Through this, the structure and operations are simplified.

What are these conditions? (next slide)
**Definition 3.7:** The Graph $G = (V,E)$ is a **Tree** if and only if

1. $G$ is loop free
2. $G$ contains no simple edged circle
3. $G$ is associating

**Counter examples**

1. loop
2. circle
3. Non-associating
Analogy

Tree as a graph

leaves
(End of the branching)

Branching
(vertices)

twigs
(edges)

root
Important: According to later implementation in Java, as a result we must consider the edges of a tree as **directed edges**.

Distinguishing the root vertex is not necessary any more.
Let $T = (V,E)$ be a tree.

- The successor of a vertex $v$ are also called *children*, or *sons* of $v$.
- The predecessor of vertex $v$ is also called *father* or *parent* of $v$.
- A vertex is the *root* of the tree when it does not have a father.
- A vertex that does not have any children is called a *leaf*.
- Vertices are called *siblings* when they have the same father.
Let $T = (V,E)$ be a tree with the root $v_0$.

- **Depth of a vertex** $v_n$
  
  = length of the path $\pi = (v_0, \ldots, v_n)$
  
  = $|\pi| = |(v_0, \ldots, v_n)|$
  
  = displacement from the root

- **Height of a tree**
  
  = maximal depth

- $T$ has the **order** $d$
  
  = the branching factor $d(T)$
  
  = Each vertex of $T$ has a maximum of $d$ children

**Nomenclature:**

- $d$-nary tree
  
  i.e.:
  
  - $d = 2$: binary tree
  
  - $d = 3$: ternary tree

![Tree Diagram]

Depth of $v_4 = 2$

Height = 3

Order = 3
In order to implement data structures based on graphs, one can use this object oriented approach.

- A vertex is an administration object with a controlled amount of child-vertices and a data object.
- An edge is created through reference to the respective child-vertices.

Tree representation

Implementation
Trees as dynamic data structures (2)

- The actual information is contained in the data object of the vertex
- The sum of all the data objects yields the amount of data that will be managed through the tree structure.

Tree structure | Amount of data
---|---
`:Node` `n_1, \ldots, n_k`
`data`

`:Node` `n_1, \ldots, n_k`
`data`

`:Object`

`:Object`

`:Object`
A binary tree represents a special case of a general tree.

- **Binary** = 2, in a pair, composed of 2 basic entities
- The structure of a binary tree and the operations associated with the tree are realized relatively easy on a digital computer through the tree characteristics.

General tree structure

Binary tree structure
Definition 3.8: The tree $T = (V,E)$ is a binary tree only if $d(T) = 2$

That means, each vertex from $T$ contains a maximum of 2 child-vertices.

examples

1. 2.
Binary trees are created in Java with the help of the BNode class.

- The **data** attribute references a data element of type **object**. (Similar to the data structure of a general tree)
- The linkage to the managing element results in the tree structure: **left** references the previous, **right** the next successor.
Binary trees: class diagrams

### BTree
- root : BNode
  - isEmpty(): boolean
  - size(): int
  - contains(Object o): boolean
  - insert( Object o): void
  - remove( Object o): void

### BNode
  - left: BNode
  - right: BNode
  - data : Object
    - getData(): Object
    - setData( Object o)

### Object
Operations of a binary tree

Entire readout
- Traversing of all vertices in a certain order.

Tree organization
- Splitting of a tree into tree parts:
  \[ \text{Tree[]} \text{ split()} \]
- Assembly of multiple trees to one new tree:
  \[ \text{Merge(Tree } t) \]

Data access
- insert: \( \text{add(Object o)} = \text{insert} \)
- delete: \( \text{remove(Object o)} = \text{delete} \)
- Search/ ask: \( \text{boolean contains(Object o)} \)
**Traversing process** is the process that runs through each vertex of a tree-forming graph exactly once. In conjunction with binary trees, one could also be talking about a **linearization**.

Prevalent traversing strategies:

**Preorder:**

„WLR“ – root, left part, right part

**Postorder:**

„LRW“ – left part, right part, root

**Inorder:**

„LWR“ – left part, root, right part
Traversing Binary trees - example

**Preorder** "WLR" yields L, B, A, D, C, E, P, N, O, R, Z

**Postorder** "LRW" yields A, C, E, D, B, O, N, Z, R, P, L

**Inorder** "LWR" yields A, B, C, D, E, L, N, O, P, R, Z
The value of the user data of all vertexes of the left part of the tree are **smaller** than the roots.

The value of the user data of all vertexes of the right part of the tree are **bigger** than the roots.