Reading for This Lecture

• Primary
  - Horowitz and Sahni, Chapter 2, Section 2
Basic Data Structures
What is a data structure?

• Data structures are schemes for organizing and storing sets.
• Data structures make it easy to perform certain set operations.
• Examples of set operations.
  – add
  – delete
  – find_min
  – delete_min
  – union
Choosing the right data structure

- Data structures consist of
  - a scheme for storing the set(s), and
  - algorithms for performing the desired operations
- Hence, each set operation has an associated complexity
- To choose a data structure, you should know
  - something about the elements of the set, and
  - what operations you will want to perform on the set.
Example: Lists

- A list is a finite sequence of elements drawn from a set
- List operations
  - `insert()`
  - `delete()`
  - `concatenate()`
  - `split()`
- List storage
  - array
  - linked list
Linked Lists

![Diagram of a linked list with nodes labeled Item 1 through Item 5. The table below shows the connections between nodes.]

<table>
<thead>
<tr>
<th>NAME</th>
<th>NEXT</th>
</tr>
</thead>
<tbody>
<tr>
<td>Empty</td>
<td>0</td>
</tr>
<tr>
<td>Item 3</td>
<td>4</td>
</tr>
<tr>
<td>Item 2</td>
<td>0</td>
</tr>
<tr>
<td>Item 1</td>
<td>3</td>
</tr>
<tr>
<td>-</td>
<td>1</td>
</tr>
</tbody>
</table>
# Linked List Operations

## INSERT

<table>
<thead>
<tr>
<th>NAME</th>
<th>NEXT</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>-</td>
</tr>
<tr>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>5</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>5</td>
<td>4</td>
</tr>
<tr>
<td>6</td>
<td></td>
</tr>
</tbody>
</table>

## DELETE

<table>
<thead>
<tr>
<th>NAME</th>
<th>NEXT</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>-</td>
</tr>
<tr>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>5</td>
<td>4</td>
</tr>
</tbody>
</table>

### INSERT

- Item 1
- Item 2
- Item 3
- Item 4
- New Item

### DELETE

- Item 1
- Item 2
- Empty
- Item 4
- Item 5
Linked List Analysis

- \texttt{make_list}(a_1, a_2, \ldots, a_n)
- \texttt{insert}(a, i)
- \texttt{delete}(i)
- \texttt{concatenate}(ptr1, ptr2)
- \texttt{split}(ptr1, i)
Data structures in algorithms

• Typically, data structures are part of a larger algorithm.
• In order to choose a data structure, you should also know something about the algorithm.
• The data structure should be efficient for the operations that will be performed most often.
• The same algorithm can have different running times using different data structures.
Arrays vs. Linked Lists

• **Linked lists**
  – Efficient to add, delete, concatenate, split.
  – Don't have to know the number of data items in advance.

• **Arrays**
  – Less storage space.
  – Fewer memory allocations.
  – More efficient to locate $i^{th}$ data item.

• **Can do hybrid schemes**
Using lists

- Insertion sort
- Merge sort/quick sort
- Binary search
- Circular lists
- Doubly linked lists
Recall: Graph consists of
- A set of *nodes* or *vertices* $V$.
- A set of *edges* $E \subseteq V \times V$.

**Adjacency matrix**
- Efficient for determining whether a particular edge is present.
- Requires $O(|V|^2)$ storage and initialization time.

**Adjacency lists**
- Usually the method of choice.
- More efficient for sparse graphs.
Stacks

- A list data structure in which insertions and deletions are made at one end is called a stack.
- This is also known as a Last In First Out (LIFO) list.
- Insert and delete operations are often called push and pop.
- Stack Data Structures
  - Array
  - Linked list
- Stacks can be used to keep track of data in recursion (stack frames).
Stack Frames

• Local data for each function call is stored on the stack.
• Each function gets a stack frame to store data.
  – space for local variables.
  – pointers to the parameters the function was called with.
  – pointer to the instruction to return to in the calling function.
  – pointer to the location to store the return value.
Queues

- A queue is a list in which insertions take place at one end and deletions at the other.
- Also known as First In First Out (FIFO) lists.
- Queue data structures
  - Array
  - Circular array
  - Linked list
Priority Queues

• A queue where each item has a specified priority.
• Additional operations for priority queues
  – find_min()
  – delete_min()
• Applications
  – sorting
  – greedy algorithms
• We will discuss these in future lectures
Graph Terminology

• Given a directed graph $G = (V, E)$, we define
  
  – a *path* is a sequence of edges $(v_1, v_2), (v_2, v_3), \ldots, (v_{n-1}, v_n)$.
  
  – such a path is said to go *from vertex* $1$ *to vertex* $n$.
  
  – A path is *simple* if no two edges on the path share a common endpoint, with the exception that we allow $v_1 = v_n$.
  
  – A simple path in which $v_1 = v_n$ is called a *cycle*.
  
  – A directed graph with no cycles is called a *directed acyclic graph*.
  
  – For vertex $w$, the number of edges $(v, w)$ in $G$ is called the *in-degree* of $w$.
  
  – Similarly for *out-degree*. 
Trees

- A (directed) tree is a directed acyclic graph satisfying the following:
  - There is exactly one vertex called the root with in-degree 0.
  - Every other vertex has in-degree 1.
  - There is a path from the root node to every other node.
- Trees also have a natural recursive definition.
- Tree terminology
  - If \((u, v) \in E\), then \(u\) is called the father/mother/parent of \(v\) and \(v\) is called the son/daughter of \(u\).
  - If there is a path from \(u\) to \(v\), then \(v\) is a descendant of \(u\) and \(u\) is an ancestor of \(v\).
More Tree Terminology

- A tree in which each node has out-degree 0, 1, or 2 is called a *binary tree*.
- A tree in which the sons are ordered is called an *ordered tree*.
- In an ordered binary tree, the two sons are usually called the *left son* and the *right son*.
- The *depth* or *level* of a vertex $v$ is the length of the (unique) path from the root to $v$.
- The depth of a tree is the maximum depth of any node.
Trees and data structures

- Trees are an element of many different data structures.
- Trees are naturally associated with recursive and divide and conquer type algorithms.
- We have already seen how trees can help us partition the elements of a set.
- Tree storage
  - arrays
  - pointers
Traversing a Tree

• Many common algorithms involve traversing or searching a tree.

• Traversal schemes
  – preorder
  – postorder
  – depth-first
  – breadth-first