Ceng 111 – Fall 2015
Week 13a

ADT

Credit: Some slides are from the “Invitation to Computer Science” book by G. M. Schneider, J. L. Gersting and some from the “Digital Design” book by M. M. Mano and M. D. Ciletti.
Other notations for computational complexity

\[ \Omega \text{ notation: } f(n) = \Omega(g(n)):\]

- Lower boundary for \( f(n) \).
- \( f(n) \) is \( \Omega(g(n)) \) iff \(|f(n)| \geq M|g(n)|\), for sufficiently large \( n \), for some constant number \( M > 0 \).

- \( 2n = \Omega(n) \)
- \( n^2 = \Omega(n^2) \)
Other notations for computational complexity

- **Θ notation**: $f(n) = \Theta(g(n))$:
  - Lower and upper boundary for $f(n)$.
  - $f(n)$ is $\Theta(g(n))$ iff $f(n)$ is bounded by below and above by $g(n)$:
    - $M|g(n)| \geq |f(n)| \geq N|g(n)|$, for sufficiently large $n$, for some constant numbers $M,N > 0$.
  - $2n = \Theta(n)$
  - $n^2 = \Theta(n^2)$
ABSTRACT DATA TYPES
Remember Stacks?

FINAL

STACK

INITIAL

Red moved to FINAL

Yellow moved to STACK

Green moved to FINAL

Yellowed moved from STACK to FINAL
Stacks

- LIFO:
  - Last In First Out
- We have seen it before (in the Shunting-Yard algorithm)
- Main operations:
  - Push
  - Pop
Stacks (cont’d)

- Operations:
  1. Push
  2. Pop
  3. Top/Peek
     - Get the top element without removing it
  4. Is-Empty
     - Checks whether the stack is empty
  5. Length
     - # of elements
Stacks in Python

Stack Operation
- Pop
- Push
- Top/Peek
- Is-Empty
- Length

Corresponding Python Op.
- L.pop()
- L.append(item)
- L[-1]
- L == []
- len(L)
Stacks in Python (Example)

- Implement postfix implementation in Python using stacks.
  - Given a string like "3 4 + 5 7 + *", evaluate and return the result.
Stacks in Python
(Example - Solution)

```python
def postfix_eval(String):
    ''' Example String: 3 4 + 5 6 * + '''
    Stack = []
    for token in String:
        if token != ' ':
            # Skip on whitespace
            if '0' <= token <= '9':
                Stack.append(token)
            else:
                operand2 = Stack.pop()
                operand1 = Stack.pop()
                result = eval(operand1 + token + operand2)
                Stack.append(str(result))  # push operation
    print Stack

Exercise: Modify it so that it works on numbers bigger than 9.
```
Stacks: Formal definition

\[ \text{push(item, stack)} \rightarrow \text{item} \odot \text{stack} \]

- \( \text{new()} \rightarrow \emptyset \)
- \( \text{popoff}(\xi \odot S) \rightarrow S \)
- \( \text{top}(\xi \odot S) \rightarrow \xi \)
- \( \text{isempty}(\emptyset) \rightarrow \text{TRUE} \)
- \( \text{isempty}(\xi \odot S) \rightarrow \text{FALSE} \)
Today

- Abstract Data Types
  - Queues
  - Trees
Queues

- **FIFO:**
  - First In First Out
  - The item that was inserted first is removed first.

- **Main operations:**
  - Add
  - Remove
Queues (cont’d)

Operations:
1. Add
2. Remove
3. Front/Peak
4. Is-Empty
5. Length
## Queues in Python

<table>
<thead>
<tr>
<th>Queue Operation</th>
<th>Corresponding Python Op.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Add</td>
<td>L.append(item)</td>
</tr>
<tr>
<td>Remove</td>
<td>L.pop(0)</td>
</tr>
<tr>
<td>Front/Peach</td>
<td>L[0]</td>
</tr>
<tr>
<td>Is-Empty</td>
<td>L == []</td>
</tr>
<tr>
<td>Length</td>
<td>len(L)</td>
</tr>
</tbody>
</table>
def bank_queue():
    Queue = []

    while True:
        if a_new_customer_arrived:
            new_customer = get_customer()
            Queue.append(new_customer)

        customer_to_be_served = Queue.pop(0)
        serve_customer(customer_to_be_served)
Queues: Formal Definition

\[ \text{add}(\text{item}, \text{queue}) \rightarrow \text{item} \uplus \text{queue} \]

- \( \text{new()} \rightarrow \emptyset \)
- \( \text{front}(\xi \uplus \emptyset) \rightarrow \xi \)
- \( \text{front}(\xi \uplus Q) \rightarrow \text{front}(Q) \)
- \( \text{remove}(\xi \uplus \emptyset) \rightarrow \emptyset \)
- \( \text{remove}(\xi \uplus Q) \rightarrow \xi \uplus \text{remove}(Q) \)
- \( \text{isempty}(\emptyset) \rightarrow \text{TRUE} \)
- \( \text{isempty}(\xi \uplus Q) \rightarrow \text{FALSE} \)
Priority Queue

- Similar to Queue except that the items in a queue has a priority value based on which they are kept in order!

- Operations:
  - `insert(item, priority)` ➔ Push item with the given priority
  - `Highest()` ➔ The item in the queue that has the highest priority
  - `Deletehighest()` ➔ Delete the item that has the highest priority
  - `Is-Empty`
  - `Length`
# Priority Queues in Python

<table>
<thead>
<tr>
<th>Priority Queue Operation</th>
<th>Corresponding Python Op.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Insert</td>
<td>L.append((item, priority))</td>
</tr>
<tr>
<td>Highest</td>
<td>Write a function that finds the max</td>
</tr>
<tr>
<td>Delete highest</td>
<td>Write a function that finds the max and deletes it</td>
</tr>
<tr>
<td>Is-Empty</td>
<td>L == []</td>
</tr>
<tr>
<td>Length</td>
<td>len(L)</td>
</tr>
</tbody>
</table>
\begin{itemize}
  \item \texttt{new()} \rightarrow \emptyset
  \item \texttt{highest}(\xi \sim \emptyset) \rightarrow \xi
  \item \texttt{highest}(\xi \sim \texttt{PQ}) \rightarrow
    \begin{tabular}{c}
    \text{if } \text{priority}(\xi) > \text{priority}(\texttt{highest}(\texttt{PQ})) \text{ then } \xi \\
    \text{else } \texttt{highest}(\texttt{PQ})
    \end{tabular}
  \item \texttt{deletehighest}(\xi \sim \emptyset) \rightarrow \emptyset
  \item \texttt{deletehighest}(\xi \sim \texttt{PQ}) \rightarrow
    \begin{tabular}{c}
    \text{if } \text{priority}(\xi) > \text{priority}(\texttt{highest}(\texttt{PQ})) \text{ then } \texttt{PQ} \\
    \text{else } \xi \sim \texttt{deletehighest}(\texttt{PQ})
    \end{tabular}
  \item \texttt{isempty}(\emptyset) \rightarrow \text{TRUE}
  \item \texttt{isempty}(\xi \sim \texttt{PQ}) \rightarrow \text{FALSE}
\end{itemize}
Trees

- Figure a: A binary tree.
- Figure b: Another binary tree.
- Figure c: A more complex tree structure.
- Figure d: A highly connected network.
- Figure e: An even more complex network.
- Figure f: A series of interconnected nodes.
- Figure g: A tree illustrating a comparison expression.
- Figure h: Another tree showing a hierarchical structure.
Example for Trees: Decision/Game Tree
Properties of Trees

- A tree is composed of nodes.
- A node can have either no branches, two branches or more than two branches.
- Binary tree: a tree where nodes have two branches.
- The depth of a tree:
  - The number of levels in the tree.
The nodes in the left branch of a node have less value than the node.

The nodes in the right branch of a node have more value than the node.
How can we represent Trees in Python?

- Nested Lists vs. Tuples / Lists
- Nested Tuples vs. Dictionaries