CSCI 104
Abstract Data Types

Mark Redekopp
David Kempe
Reviewed for Fall 2022
Abstract Data Types

- An **abstract data type**, or ADT, as a specification or model for a group of values/data and the operations on those values
- A **data structure** is a specific implementation of an ADT in a given programming language
- As an analogy think of the ADT as the class declaration (header file) and the data structures are various implementations of a specific ADT (source files)
- Given an application we can quickly identify the ADT and then proceed to choose an appropriate data structure
- Each data structure we will examine in this course has certain:
  - Well defined operations and capabilities that are often useful
  - Time & space advantages
  - Time & space disadvantages
- You need to know those operations, advantages and disadvantages
Popular ADTs

• The "Big 3" ADTs
  – List
    • 3 specialized List ADTs: Queues, Stacks, Deques
  – Set
  – Map (Dictionary)

• Other ADTs
  – Priority Queue
  – Graphs
Lists

• Ordered collection of items, which may contain duplicate values, usually accessed based on their position (index)
  – Ordered = Each item has an index and there is a front and back (start and end)
  – Duplicates allowed (i.e. in a list of integers, the value 0 could appear multiple times)
  – Accessed based on their position (list[0], list[1], etc.)
• What are some operations you perform on a list?
# List Operations

<table>
<thead>
<tr>
<th>Operation</th>
<th>Description</th>
<th>Input(s)</th>
<th>Output(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>insert</td>
<td>Add a new value at a particular location shifting others back</td>
<td>Index : int Value</td>
<td></td>
</tr>
<tr>
<td>remove</td>
<td>Remove value at the given location</td>
<td>Index : int</td>
<td>Value at location</td>
</tr>
<tr>
<td>get / at</td>
<td>Get value at given location</td>
<td>Index : int</td>
<td>Value at location</td>
</tr>
<tr>
<td>set</td>
<td>Changes the value at a given location</td>
<td>Index : int Value</td>
<td></td>
</tr>
<tr>
<td>empty</td>
<td>Returns true if there are no values in the list</td>
<td></td>
<td>bool</td>
</tr>
<tr>
<td>size</td>
<td>Returns the number of values in the list</td>
<td></td>
<td>int</td>
</tr>
<tr>
<td>push_back / append</td>
<td>Add a new value to the end of the list</td>
<td>Value</td>
<td></td>
</tr>
<tr>
<td>find</td>
<td>Return the location of a given value</td>
<td>Value</td>
<td>Int : Index</td>
</tr>
</tbody>
</table>

© 2022 by Mark Redekopp. This content is protected and may not be shared, uploaded, or distributed.
Queues and Stacks

• Two specialized List ADTs

**Queue**
- Items enter from one side (often the back...push_back)
- Items leave from the other side (often the front...pop_front)

**Stack**
- Items enter and leave from the same side (i.e. the top)
- (push)
- (pop)
# Queue & Stack Operations

<table>
<thead>
<tr>
<th>Queues</th>
<th>Operations Relative to Lists</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>insert</td>
<td></td>
<td></td>
</tr>
<tr>
<td>remove</td>
<td></td>
<td></td>
</tr>
<tr>
<td>get (front)</td>
<td></td>
<td>Can only get front item</td>
</tr>
<tr>
<td>set</td>
<td></td>
<td></td>
</tr>
<tr>
<td>empty</td>
<td></td>
<td></td>
</tr>
<tr>
<td>size</td>
<td></td>
<td></td>
</tr>
<tr>
<td>push_back</td>
<td></td>
<td>Add to one side</td>
</tr>
<tr>
<td>pop_front</td>
<td></td>
<td>Remove from the other</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Stacks</th>
<th>Operations Relative to Lists</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>insert</td>
<td></td>
<td></td>
</tr>
<tr>
<td>remove</td>
<td></td>
<td></td>
</tr>
<tr>
<td>get (top)</td>
<td></td>
<td>Can only get top item</td>
</tr>
<tr>
<td>set</td>
<td></td>
<td></td>
</tr>
<tr>
<td>empty</td>
<td></td>
<td></td>
</tr>
<tr>
<td>size</td>
<td></td>
<td></td>
</tr>
<tr>
<td>push</td>
<td></td>
<td>Add to one side</td>
</tr>
<tr>
<td>pop</td>
<td></td>
<td>Remove from the same</td>
</tr>
</tbody>
</table>

© 2022 by Mark Redekopp. This content is protected and may not be shared, uploaded, or distributed.
Maps / Dictionaries

• Stores **key, value pairs**
  – Example: Map student names to their GPA
• Keys must be **unique** (can only occur once in the structure)
• No constraints on the values (can have duplicates)
• What operations do you perform on a map/dictionary?
  • No inherent ordering between key, value pairs
    – Can't ask for the 0\text{th} item...
• **Primary operations:**
  – Insert, remove, find/lookup

© 2022 by Mark Redekopp. This content is protected and may not be shared, uploaded, or distributed.
# Map / Dictionary Operations

<table>
<thead>
<tr>
<th>Operation</th>
<th>Description</th>
<th>Input(s)</th>
<th>Output(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Insert / add</td>
<td>Add a new key,value pair to the dictionary (assuming it's not there already)</td>
<td>Key, Value</td>
<td></td>
</tr>
<tr>
<td>Remove</td>
<td>Remove the key,value pair with the given key</td>
<td>Key</td>
<td></td>
</tr>
<tr>
<td>Get / lookup</td>
<td>Lookup the value associated with the given key or indicate the key,value pair doesn't exist</td>
<td>Key</td>
<td>Value associated with the key</td>
</tr>
<tr>
<td>In / Find</td>
<td>Check if the given key is present in the map</td>
<td>Key</td>
<td>bool (or ptr to pair/NULL)</td>
</tr>
<tr>
<td>empty</td>
<td>Returns true if there are no values in the list</td>
<td></td>
<td>bool</td>
</tr>
<tr>
<td>size</td>
<td>Returns the number of values in the list</td>
<td></td>
<td>int</td>
</tr>
</tbody>
</table>
Set

- A set is a dictionary where we only store keys (no associated values)
  - Example: All the courses taught at USC (ARLT 100, ..., CSCI 104, MATH 226, ...)
- Items (a.k.a. Keys) must be **unique**
  - No duplicate keys (only one occurrence)
- Not accessed based on index but on value
  - We wouldn't say, "What is the 0th course at USC?"
- What operations do we perform on a set?
  - Similar to a map
  - Insert, remove, find/in
## Set Operations

<table>
<thead>
<tr>
<th>Operation</th>
<th>Description</th>
<th>Input(s)</th>
<th>Output(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Insert / add</td>
<td>Add a new key to the set (assuming its not there already)</td>
<td>Key</td>
<td></td>
</tr>
<tr>
<td>Remove</td>
<td>Remove</td>
<td>Key</td>
<td></td>
</tr>
<tr>
<td>In / Find</td>
<td>Check if the given key is present in the map</td>
<td>Key</td>
<td>bool (or ptr to item/NULL)</td>
</tr>
<tr>
<td>empty</td>
<td>Returns true if there are no values in the list</td>
<td></td>
<td>bool</td>
</tr>
<tr>
<td>size</td>
<td>Returns the number of values in the list</td>
<td></td>
<td>Int</td>
</tr>
<tr>
<td>intersection</td>
<td>Returns a new set with the common elements of the two input sets</td>
<td>Set1, Set2</td>
<td>New set with all elements that appear in both set1 and set2</td>
</tr>
<tr>
<td>union</td>
<td>Returns a new set with all the items that appear in either set</td>
<td>Set1, Set2</td>
<td>New set with all elements that appear in either set1 and set2</td>
</tr>
<tr>
<td>difference</td>
<td>Returns a set with all items that are just in set1 but not set2</td>
<td>Set1, Set2</td>
<td>New set with only the items in set1 that are not in set2</td>
</tr>
</tbody>
</table>
Intersection, Union, Difference

• May be familiar from CS 170

• Set intersection
  – $S_1 \cap S_2$

• Set Union
  – $S_1 \cup S_2$

• Set Difference
  – $S_1 – S_2$
What's Your ADT?

- Scores on a test
- Students in a class
- Courses & their enrollment
- Temperature Reading at a location
- Usernames and password
- Index in a textbook
- Facebook friends

- List
- Set (maybe List)
- Map (Key = course, Value = enrollment)
- List
- Map
- Map
- Set
Some Implementation Details

• **List**
  – An array acts as a list
  – Index provides ordering
    • First at location 0
    • Last at location n-1

• **Set**
  – Can use an array
  – Must check for duplicate on insertion
    • O(n) solution
  – Can we do better? Yes...

• **Map**
  – Can also use an array
  – Again check for duplicate key on insertion

```cpp
struct Pair{
    string key;
    double value;
};
```
Priority Queue ADT

• Operations
  – Can add items in any order
  – Only allows retrieval of the "best/top" priority item (however "best" is defined: smallest, largest, etc.)
  – Only allows removal of the "best/top" item

• Can be stored as a "sorted" list
  – But there are more efficient implementations
Graph ADT

• Stores nodes (aka vertices) and edges between the nodes
  – Edges model relationships between vertices
  – Note: a "tree" is common form of a graph

• Can be stored as a list of lists or a __________