Data Abstraction and Basic Data Structures

- Improving efficiency by building better Data Structure
- Object IN
  - Abstract Data Type
    - Specification
    - Design
  - Architecture [Structure, Function]
- Abstract Data Types
  - Lists, Trees
  - Stacks, Queues
  - Priority Queue, Union-Find
  - Dictionary

Abstract Data Type

- i is an instance of type T, i ∈ T
- e is an element of set S, e ∈ S
- o is an object of class C, o ∈ C
- Abstract Data Type
  - Structures: data structure declarations
  - Functions: operation definitions
- An ADT is identified as a Class
  in languages such as C++ and Java
- Designing algorithms and proving correctness of algorithms
  - based on ADT operations and specifications

ADT Specification

- The specification of an ADT describe how the operations (functions, procedures, or methods) behave
  - in terms of Inputs and Outputs
- A specification of an operation consists of:
  - Calling prototype
  - Preconditions
  - Postconditions
- The calling prototype includes
  - name of the operation
  - parameters and their types
  - return value and its types
- The preconditions are statements
  - assumed to be true when the operation is called.
- The postconditions are statements
  - assumed to be true when the operation returns.

Operations for ADT

- Constructors
  - create a new object and return a reference to it
- Access functions
  - return information about an object, but do not modify it
- Manipulation procedures
  - modify an object, but do not return information
- State of an object
  - current values of its data
- Describing constructors and manipulation procedures
  - in terms of Access functions
- Recursive ADT
  - if any of its access functions returns the same class as the ADT

ADT Design e.g. Lists

- Every computable function can be computed using Lists as the only data structure!
- IntList cons(int newElement, IntList oldList)
  - Precondition: None.
  - Postconditions: If x = cons(newElement, oldList) then
    1. x refers to a newly created object;
    2. x != nil;
    3. first(x) = newElement;
    4. rest(x) = oldList
- int first(IntList aList) // access function
  - Preconditions: aList != nil
- IntList rest(IntList aList) // access function
  - Preconditions: aList != nil
- IntList nil //constant denoting the empty list.

Binary Tree

- A binary tree T is a set of elements, called nodes, that is empty or satisfies:
  1. There is a distinguished node r called the root
  2. The remaining nodes are divided into two disjoint subsets, L and R, each of which is a binary tree.
     - L is called the left subtree of T and R is called the right subtree of T.
- There are at most 2^d nodes at depth d of a binary tree.
- A binary tree with n nodes has height at least Ceiling[lg(n+1)] – 1.
- A binary tree with height h has at most 2^{h+1} – 1 nodes
### Stacks
- A stack is a linear structure in which insertions and deletions are always made at one end, called the top.
- This updating policy is called last in, first out (LIFO)

### Queue
- A queue is a linear structure in which
  - all insertions are done at one end, called the rear or back, and
  - all deletions are done at the other end, called the front.
- This updating policy is called first in, first out (FIFO)

### Priority Queue
- A priority queue is a structure with some aspects of FIFO queue but
  - in which element order is related to each element’s priority,
  - rather than its chronological arrival time.
- As each element is inserted into a priority queue, conceptually it is inserted in order of its priority.
- The one element that can be inspected and removed is the most important element currently in the priority queue.
  - a cost viewpoint: the smallest priority
  - a profit viewpoint: the largest priority

### Union-Find ADT for Disjoint Sets
- Through a Union operation, two (disjoint) sets can be combined.
  - to insure the disjoint property of all existing sets, the original two sets are removed and the new set is added
  - Let the set id of the original two set be, s and t, s != t
  - Then, new set has one unique set id that is either s or t.
- Through a Find operation, the current set id of an element can be retrieved.
  - Often elements are integers and
  - the set id is some particular element in the set, called the leader, as in the next e.g.

### Dictionary ADT
- A dictionary is a general associative storage structure.
- Items in a dictionary
  - have an identifier, and
  - associated information that needs to be stored and retrieved.
- no order implied for identifiers in a dictionary ADT

---

**Union-Find ADT e.g.**
- UnionFind create(int n)
  - // create a set (called sets) of n singleton disjoint sets
    - {{1},{2},{3},..., {n}}
  - int find(UnionFind sets, c)
    - // return the set id for c
  - void makeSet(unionFind sets, int c)
    - // union one singleton set {c} (c not already in the sets)
      - into the exiting sets
  - void union(UnionFind sets, int s, int t)
    - // s and t are set ids, s != t
    - // a new set is created by union of set [s] and set [t]
    - // the new set id is either s or t, in some case min(s, t)

**Dictionary ADT**
- A dictionary is a general associative storage structure.
- Items in a dictionary
  - have an identifier, and
  - associated information that needs to be stored and retrieved.
- no order implied for identifiers in a dictionary ADT