14 - Set Data Structures

Joseph Afework
CS 241

Dept. of Computer Science
California Polytechnic State University, Pomona, CA

Agenda

- Intro
- Operations
- Implementation
- Performance

Reading Assignment

- Read Chapter 28
  - Chapter 28 (Read about: **Sets**)

Sets

- A **set** is an unordered collection of objects.
- Builds on the mathematical concepts: ***(Remember CS 130)***
  - **Union**
  - **Subtraction**
  - **Difference (Subtraction)**
  - **Subset**

Set Rules

Rules:

1. Elements cannot be repeated (**unique**)
2. Elements in the set usually share some sort of logical grouping **(organization)**.

Example

- Students at Cal Poly Pomona
  - Unique members?
  - Logical organization?
- Students currently taking CS 241
  - Unique members?
  - Logical organization?

Empty Set

- If it makes sense for a set to contain 0 members, it is said to be an **empty set** or **null set**.

- **Example:**
    - If CS 241 is not offered, then the set won't contain any members.

Set Operations

1. Union
2. Subtraction
3. Difference (Subtraction)
4. Subset

Note: *Behavior is just like the mathematical definition of sets...*

Union

Union: Combine two or more sets into a new set that contains all of the values from the original sets in the new set.

$$A \cup B = \{x : x \in A \text{ or } x \in B\}$$

Note: *Generally duplicates are ignored (include only 1 copy in the new set)*

Intersection

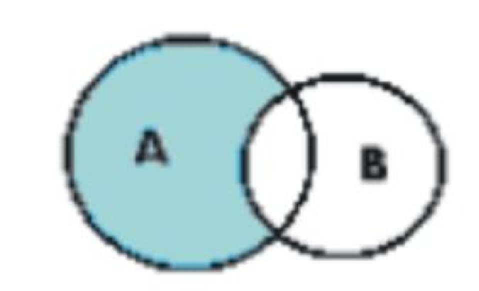
Intersection: Construct a new set with only the elements common to the sets being evaluated.

$$A \cap B = \{x : x \in A \land x \in B\}$$

Note: *Generally duplicates are ignored (include only 1 copy in the new set)*

Difference (Subtraction)

Difference: Given two sets, A and B, construct a new set C which contains the elements in Set A that do not exist in Set B.



Note: *Generally duplicates are ignored (include only 1 copy in the new set)*

Subset

Subset: Given two sets, A and B, determine if all of the elements in A are already present in B. If so, then A is a subset of B.

$$A \subseteq B$$

Note: *Generally duplicates are ignored (include only 1 copy in the new set)*

Implementation

Like more ADT, it is possible to implement a set data structure using different data structures.

1. Use an **array** or array list
2. Use a **tree**
3. And more...

Array Based Set: Insertion

Insert(Set A, element B):

Loop through elements in A (let e = element)

If **e equals B**

return *// Element exists no need to insert*

A[i] = B;

Runtime: O(speed of lookup) …. Speed of look up in an array is O(n)... **O(n)**

Array Based Set: Search

Search(Set A, element B):

Loop through elements in A (let e = element)

If **e equals B**

return *// Element found*

return null // Element not found

Runtime: O(speed of lookup) …. Speed of look up in an array is O(n)... **O(n)**

Array Based Set: Delete

```
Delete(Set A, element B):
    Loop through elements in A (let e = element)
        If e equals B
            Int i = indexOf(e)
            A[i] = null
            Swap last element with A[i] // move null to end of array
            return // Element removed
    return null // Element not found
```

Runtime: O(speed of lookup) …. Speed of look up in an array is O(n)... **O(n)**

BONUS - Array Based Set: Union

Union(Set A, Set B):

Set C = Set A

Loop through elements in B (let e = element)

If **e does not exist in A**, add it to set C

Return set C

Runtime: O(n * speed of lookup) …. Speed of look up in an array is O(n)... $O(n^2)$

Array Based Set Performance

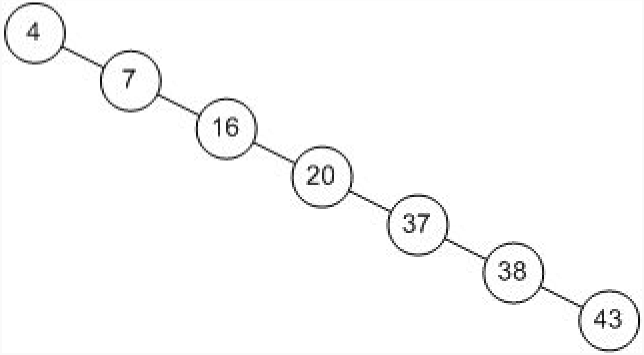
Can we do better?

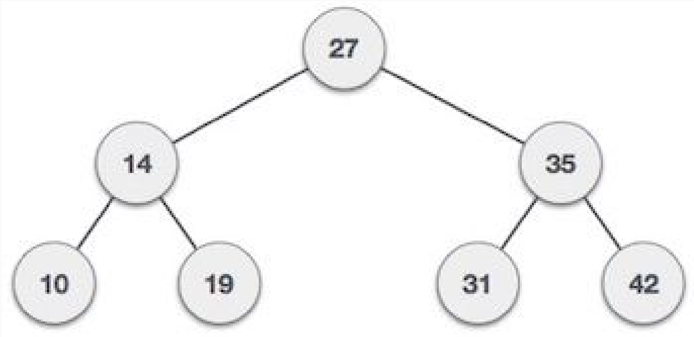
Set (Array)	Worst Case
Insert	O(n)
Delete	O(n)
Search	O(n)
Union	$O(n^2)$

Remember

Self Balancing Trees (Faster lookup)?

- **AVL Tree**
- Red-Black Tree





AVL Based Set: Insertion

Insert(Set A, element B):

Insert B into tree using AVL rules, ignore if value already exists.

Runtime: O(speed of insertion) …. Speed of AVL insertion is **O(log(n))**

AVL Based Set: Search

Search(Set A, element B):

Perform Binary Search in AVL tree

Runtime: O(speed of search) …. Speed of search is **O(log(n))**

AVL Based Set: Delete

Delete(Set A, element B):

Delete from AVL tree using AVL rules

Runtime: O(speed of deletion) …. Speed of look up in an array is O(n)... **O(log(n))**

BONUS - AVL Based Set: Union

Union(Set A, Set B):

Set C = Set A

Loop through elements in B (traversal)

Insert B into tree using AVL rules, ignore if value already exists.

Return set C

Runtime: O(n * speed of insertion) …. Speed of AVL insertion is O(log(n))... **O(n * log(n))**

AVL Based Set Performance

Set (AVL)	Worst Case
Insert	O(log(n))
Delete	O(log(n))
Search	O(log(n))
Union	O(n*log(n))

Array Backed Set vs AVL Backed Set

Set	(Array) Worst Case	(AVL) Worst Case
Insert	O(n)	O(log(n))
Delete	O(n)	O(log(n))
Search	O(n)	O(log(n))
Union	$O(n^2)$	O(n*log(n))

References

https://www.slideshare.net/Tech_MX/set-data-structure-i

http://www.cs.bham.ac.uk/research/projects/poplog/paradigms_lectures/lecture16.html

http://blog.benoitvallon.com/data-structures-in-javascript/the-set-data-structure/

https://msdn.microsoft.com/en-us/library/aa289153(v=vs.71).aspx

http://web.cs.wpi.edu/~cs2102/common/notes-d13/bsts-and-avls.html