Name : Mu In Nasif
Roll : 001910501036
Class : BCSE II
Sem : 3
Session : 2020-2021

Data Structures and Algorithms Assignment Set 2

Question - 4: Define an ADT for Set. Write C data representation and functions for the operations on the Set in a Header file, with array as the base data structure. Write a menu-driven main program in a separate file for testing the different operations and include the above header file.

Solution Approach :
We will implement sets to represent mathematical (finite) sets as closely as possible, with the following properties:
1. Sets are collections of similar objects called elements (with no particular order) and no two elements in a set are same.
2. (Finite) sets have a cardinality which is basically the number of elements in the set.
3. We can check for membership (containment) of elements within a set.
4. Union, intersection and subtraction can be done on two (or more than two) sets.
5. We can also check for subsets (whether one set is a subset/proper subset of the other) and whether two sets are disjoint; we can also check for null sets. In addition to the above, we will also be adding support for addition and removal of elements in our sets. We will also need support for iterating/traversing over the elements in the set (the order of the elements is irrelevant). Since the problem requires we use an array as a backing data structure, we will use array lists/vectors developed in problem 3 of this assignment for the implementation

Pseudocode:

Assume the following operations are defined for lists (which are array lists, we use the following as general so alternative implementations can also be easily switched in):

list_length(list) to obtain the length of list
list_search(list, element) to search for element in list, true if present and false otherwise
list_insert(list, element) to add an element in the list (location is irrelevant)
list_remove(list, element) to remove an element in the list (location is irrelevant)
And also the list object must have the property that we can traverse/iterate over the elements in the list.

This traversal operation can be used as-is for sets.

```
set_cardinality(S) → list_length(S) //number of elements in set S the list length
set_is_null(S) → set_cardinality(S) == 0 //Null sets have cardinality 0
set_is_member(S, x) → list_search(S, x) //Search in set S for checking memebership of x in S


set_add_member(S, x) {//Add a unique element x into the set
    if set_is_member(S, x) is true: return with status indicating x was already present
    list_insert(S, x)
    return with status indicating x was added
}

set_remove_member(S, x) → list_remove(S, x) //Remove from set = Remove from list

set_union(A, B) {//Return union of sets A and B
    Create C as a new empty set/list
    for each x in A: set_add_member(C, x)
    for each x in B: set_add_member(C, x) //set_add_member takes care of common
    elements
    return C
}

set_intersection(A, B) {
    //Return intersection of A and B Create C as a new empty set/list
    for each x in A {
        if set_is_member(B, x) is true: set_add_member(C, x) //Add iff present in both A
        and B
    }
    return C
}

set_subtraction(A, B) {
    //Subtract B from A and return result Create C as a new empty set/list
    for each x in A {
        if set_is_member(B, x) is false: set_add_member(C, x) //Add iff present in A but
        not in B
    }
    return C
}
```

```
set_is_subset(A, B) {
    //Check if A is a subset of B
    for each x in A {
        if set_is_member(B, x) is false: return false //Not a subset if at least one element
        of A is not present in B
    }
    return true //All elements of A are in B
}

set_is_proper_subset(A, B) → set_is_subset(A, B) and set_cardinality(A) < set_cardinality(B)
/*A is a proper subset of B if A is a subset of B  and if there is at least one element of B not in A;
or in case of finite sets, cardinality of B is more than that of A*/

set_is_disjoint(A, B) {
    //Check if A and B have no elements common
    for each x in A {
        if set_is_member(B, x) is true: return false //Not disjoint if at least one element of
        A is common to B
    }
    return true //All elements of A are not in B
}
```

Code: a2q3_vector_nonsentinel.h was borrowed from problem 3 of this assignment to provide the backing data structure for the set. This header file is used by p4_set.h which contains the structure definition and operations for implementing sets as described above. Lastly a2q4_test.c is a menu-driven program to test the header files. All source code conforms to ANSI C89