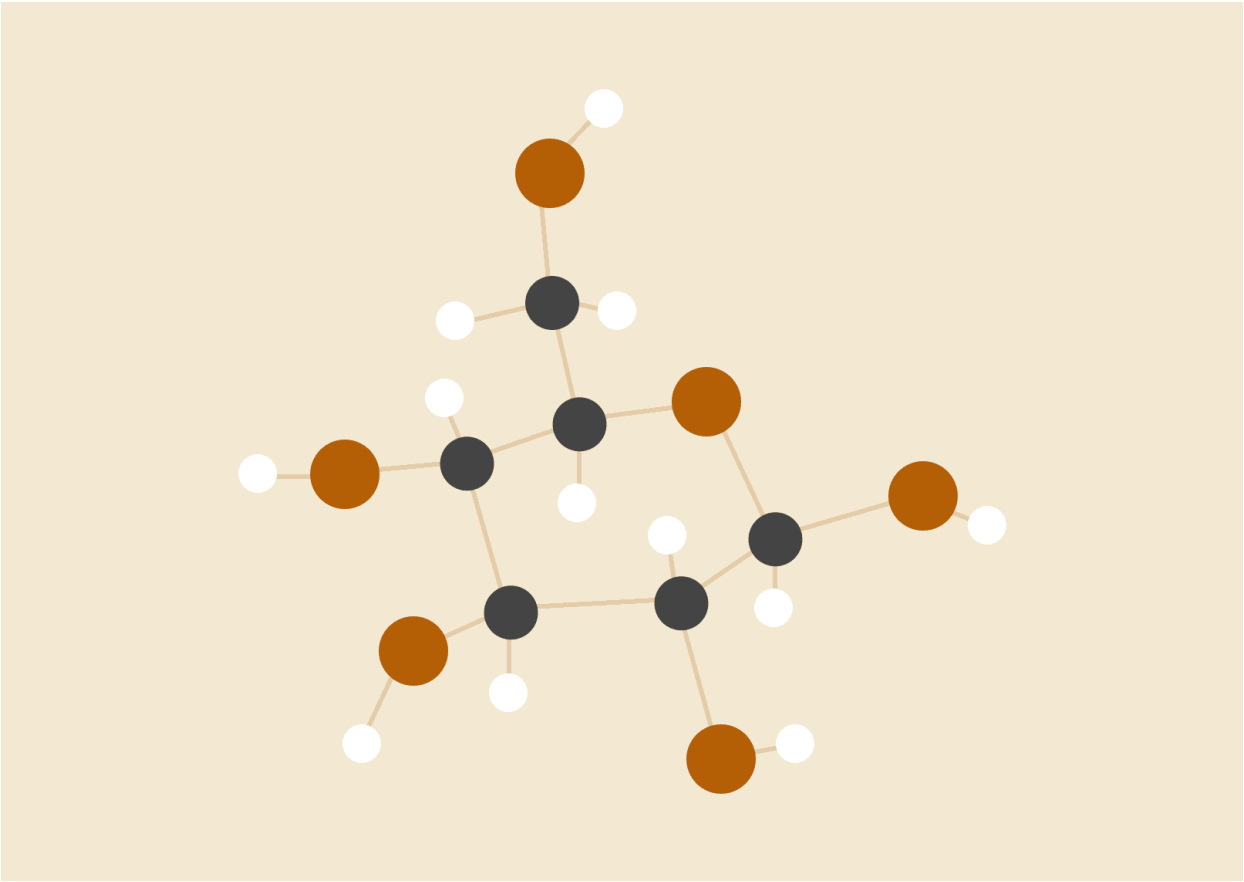
DSA LAB REPORT

CLASS *BCSE II* ***SEM*** *FIRST* ***SESSION*** *2020-21*



NAME Neeladri Pal

ROLL 001910501015

ASSIGNMENT SET - 2

PROBLEM NUMBER - 1

PROBLEM STATEMENT

Define an ADT for Polynomials.

Write C data structure representation and functions for the operations on the Polynomials in a Header file.

Write a menu-driven main program in a separate file for testing the different operations and include the above header file.

SOLUTION APPROACH

ADT Polynomial

objects: $p(x) = a_1x^{e1} + a_2x^{e2} + ... + a_nx^{en}$; a set of ordered pairs of $<a_i, e_i>$ where a_i in *Coefficients* and e_i in *Exponents*, e_i are integers >= 0

functions: for all *poly*, *poly*1, *poly*2 ∈ *Polynomial*, *coeff* ∈ *Coefficients*, *expo* ∈ *Exponents*, *cnst* ∈ *Number*

1. *Boolean* IsZero (*poly*) ::= **if** (*poly*) **return** TRUE **else return** FALSE
2. *Coefficient* Coef (*poly*, *expo*) ::= **if** (*expo* ∈ *poly*) **return** its coefficient **else return** 0
3. *Exponent* Degree (*poly*) ::= **return** the largest exponent in *poly*
4. Attach (*poly, coeff, expo*) ::= append a new pair <*coeff*, *expo*> to *poly*
5. AddSingleTerm (*poly, coeff, expo*) ::= **if** (*expo* ∈ *poly*) update coefficient of the pair **else** append new pair <*coef*, *expo*> to *poly*
6. CMult (*poly, const*) ::= multiply coefficient of every pair in *poly* by *cnst*
7. Add (*poly, poly1, poly2*) ::= compute the polynomial *poly* = *poly1* + *poly2*
8. Mult (*poly, poly1, poly2*) ::= compute the polynomial *poly* = *poly1* · *poly2*

Representation We store all the polynomials in a single *terms* array. Each polynomial is represented by a <*start*, *finish*> pair, where start, finish are integers >= 0, and denote the start and end indices of the terms of the polynomial in the *terms* array. If a polynomial has n non-zero terms *finish* = *start* + n - 1.

STRUCTURED PSEUDOCODE

```
MAX_TERMS = 1000

structure polterm :
        coeff
        expo

array polterm terms [MAX_TERMS]
terms_avail = 0

structure poly :
        start
        finish

function isZero (poly A) :
        if A.finish == A.start - 1
                return TRUE
        else
                return FALSE

function coef (poly A, e) :
        for i = A.start to A.finish in steps of 1
                if terms[i].expo == e
                        return terms[i].coeff
                end if
        end for
        return 0

function degree (poly A) :
        if A is a Zero Polynomial
                return 0
        else
                return terms[A.start].expo
        end if
function  attach (poly* X, c, e) :
        if c is non-zero
                if total number of terms > MAX_TERMS
                        print Error Message
```

```
                    return
                end if
                terms [terms_avail].coeff = c
                terms [terms_avail].expo = e
                terms_avail = terms_avail + 1
                X -> finish = terms_avail - 1
        end if

function  addSingleTerm (poly* X, c, e) :
        if c is non-zero
                for i = X -> start to X -> finish in steps of 1
                        if a term exists with exponent e
                                term[i].coeff = term[i].coeff + c
                                return
                        end if
                end for
                attach (X, c, e)
        end if

function add (poly* X, poly A, poly B) :
        X -> start = terms_avail
        X -> finish = terms_avail - 1
        i = A.start, j = B.start
        while i <= A.finish and j <= B.finish
                if terms[i].expo == terms[j].expo
                        c = terms[i].coeff + terms[j].coeff
                        attach (X, c, terms[i].expo),
                        i = i + 1
                        j = j + 1
                else if terms[i].expo > terms[j].expo
                        attach ith term of A to X
                        i = i + 1
                else
                        attach jth term of B to X
                        j = j + 1
                end if
        end while
```

```
        if some terms of A or B are still left to be encountered
                attach these terms to X

function cMult (poly A, m) :
        for i = A.start to A.finish in steps of 1
                multiply coefficient of ith term by m
        end for

function mult (poly* X, poly A, poly B) :
        X -> start = terms_avail
        X -> finish = terms_avail - 1
        for i = A.start to  A.finish in steps of 1
                for j = B.start to B.finish in steps of 1
                        c = terms[i].coeff * terms[j].coeff
                        e = terms[i].expo + terms[j].expo
                        attach (X, c, e)
                end for
        end for
        sort the terms array from X.start to X.finish in decreasing order
```

RESULTS

```
neeladripal@Neeladris-Macbook-Air Assignment-Set-II-Solutions % gcc -o a1 sol1.c
neeladripal@Neeladris-Macbook-Air Assignment-Set-II-Solutions % ./a1

Polynomial Operations -->
1. Check if a polynomial is Zero polynomial
2. Find coefficient of an exponent
3. Multiply polynomial by a constant
4. Display degree of a polynomial
5. Add two polynomials
6. Multiply two polynomials
Enter your choice: 1

Enter a polynomial -->
Enter number of terms: 2
Enter terms <coeff-i, exp-i>:
0 6
0 7

Zero Polynomial.
Do you want to continue (y/n)? y
```

```
Polynomial Operations -->
1. Check if a polynomial is Zero polynomial
2. Find coefficient of an exponent
3. Multiply polynomial by a constant
4. Display degree of a polynomial
5. Add two polynomials
6. Multiply two polynomials
Enter your choice: 2

Enter a polynomial -->
Enter number of terms: 2
Enter terms <coeff-i, exp-i>:
3 4
5 1

Enter exponent: 4

Coefficient of x^4 = 3.000000
Do you want to continue (y/n)? y

Polynomial Operations -->
1. Check if a polynomial is Zero polynomial
2. Find coefficient of an exponent
3. Multiply polynomial by a constant
4. Display degree of a polynomial
5. Add two polynomials
6. Multiply two polynomials
Enter your choice: 3

Enter a polynomial -->
Enter number of terms: 3
Enter terms <coeff-i, exp-i>:
4 5
3 2
1 0

Enter constant: 4

Resultant Polynomial: 16.000x^5 + 12.000x^2 + 4.000

Do you want to continue (y/n)? y

Polynomial Operations -->
1. Check if a polynomial is Zero polynomial
2. Find coefficient of an exponent
3. Multiply polynomial by a constant
4. Display degree of a polynomial
5. Add two polynomials
6. Multiply two polynomials
Enter your choice: 4

Enter a polynomial -->
Enter number of terms: 3
Enter terms <coeff-i, exp-i>:
5 3
2 6
1 0

Degree: 6
```

```
Do you want to continue (y/n)? y

Polynomial Operations -->
1. Check if a polynomial is Zero polynomial
2. Find coefficient of an exponent
3. Multiply polynomial by a constant
4. Display degree of a polynomial
5. Add two polynomials
6. Multiply two polynomials
Enter your choice: 5

Enter a polynomial -->
Enter number of terms: 3
Enter terms <coeff-i, exp-i>:
4 2
7 0
2 4

Enter second polynomial -->
Enter number of terms: 2
Enter terms <coeff-i, exp-i>:
6 2
3 1

Resultant Polynomial: 2.000x^4 + 10.000x^2 + 3.000x^1 + 7.000

Do you want to continue (y/n)? y

Polynomial Operations -->
1. Check if a polynomial is Zero polynomial
2. Find coefficient of an exponent
3. Multiply polynomial by a constant
4. Display degree of a polynomial
5. Add two polynomials
6. Multiply two polynomials
Enter your choice: 6

Enter a polynomial -->
Enter number of terms: 3
Enter terms <coeff-i, exp-i>:
4 3
1 1
5 0

Enter second polynomial -->
Enter number of terms: 2
Enter terms <coeff-i, exp-i>:
7 4
7 1

Resultant Polynomial: 28.000x^7 + 7.000x^5 + 63.000x^4 + 7.000x^2 + 35.000x^1
```

DISCUSSION

When a new polynomial is created, its start index is equal to the terms_avail, i.e., the next free slot available for a polynomial to store value. New terms are added if exponent >= 0 and coefficient is non-zero. On adding a term, the finish index becomes equal to terms_avail and then terms_avail is incremented by 1. Terms can be added as long as terms_avail does not exceed MAX_TERMS. Also, the terms of a polynomial are sorted in the decreasing order of their exponents.
Functions →

1. IsZero (*poly* A) - returns a value based on a condition; works in O(1) time complexity.
2. Coef (*poly* A, *e*) - traverses the terms array to find the coefficient of the term with the given exponent; works in O(n) time complexity.
3. Degree (*poly* A) - returns the exponent of the term in terms array with index = start index of the polynomial; works in O(1) time complexity.
4. Attach (*poly** X, c, e) - appends a new {coefficient, index} element to the terms array; works in O(1) time complexity.
5. AddSingleTerm (*poly** X, c, e) - traverses the array to find if a term with the given exponent exists which takes O(n) time; if match found, updates the coefficient of the term, else appends the new {coefficient, index} element to the terms array, both of which takes O(1) time; overall time complexity is O(n)
6. CMult (*poly* A, c) - multiplies coefficient of every element in the polynomial by a constant; works in O(n) time complexity
7. Add (*poly** X, *poly* A, *poly* B) -if polynomial A has m elements and B has n elements, a term in polynomial X is formed either by adding coefficients of terms in A and B having common exponents or by copying the term from A or B. This operation proceeds in such a manner that the resultant polynomial always has its terms sorted in decreasing order of their exponents. Thus the worst time complexity is O(m + n).
8. Mult (*poly** X, *poly* A, *poly* B) - every term of polynomial A is multiplied with every term of polynomial B which takes O(mn). For each term multiplication operation, before adding the new term, the previously computed terms are searched for a common exponent which has O(m+n) worst time complexity, as there can be at most m+n elements after multiplication. Therefore, overall time complexity is $O(m \cdot n \cdot (m+n))$.

SOURCE CODE

polynomial.h, sol1.c

PROBLEM NUMBER - 2

PROBLEM STATEMENT

Define an ADT for Sparse Matrix.

Write C data structure representation and functions for the operations on the Sparse Matrix in a Header file.

Write a menu-driven main program in a separate file for testing the different operations and include the above header file.

SOLUTION APPROACH

ADT SparseMatrix

objects: A sequence or a chain of elements of the form <*row, col, val*> where *row* denotes row number , *col* denotes column number and *val* denotes value (must be non-zero) in the cell. Usually in a sparse matrix, the number of non zero entries is less than the number of zero entries.

functions: for all *a,b* ∈ *SparseMatrix*, *i, j, maxCol, maxRow* ∈ *index*

1. *SparseMatrix* Create *(maxRow, maxCol)* ::= **return** a sparse matrix that can hold upto maxItems = *maxRow* * *maxCol* and whose maximum row size is *maxRow* and maximum column size is *maxCol*.
2. *SparseMatrix* Transpose (*a*) ::= **return** a sparse matrix generated by interchanging the row and column of the elements of *a*.
3. *SparseMatrix* add (*a, b*) ::= **if** number of rows and columns of *a* and *b* are equal **return** the sparse matrix obtained by adding elements with same row and column value **else return** error
4. *SparseMatrix* mult (*a, b*) ::= **if** number of columns in a equals the number of rows in a, **return** the sparse matrix c obtained as c[i][j] = sum of all A[i][k] * B[k][j] (for 0 <= k <= number of columns in a), **else return** error

Representation We form a structure *sparseterm* which contains row, column, value and a sparse matrix is a set of elements of type *sparseterm*. The first element contains the number of rows, number of columns, and number of non-zero terms as <*row, column, value*>.

STRUCTURED PSEUDOCODE

```
MAX_TERMS = 101

structure sparseterm :
        row, col, value

function createMatrix (maxRow, maxCol) :
        s = sparseterm array of size MAX_TERMS
        input n = number of non-zero terms
        if n > maxRow * maxCol
                print Error Message
                return
        else
                s[0].row = maxRow, s[0].col = maxCol, s[0].value = n
                for i = 1 to n in steps of 1
                        input s[i].row, s[i].col, s[i].value
                        if index out of range
                                print Error Message
                return s

function transpose (sparseterm array A) :
        rowTerms, startPos = arrays of size :number of columns in A
        B = sparseterm array of size MAX_TERMS
        between A and B, interchange number of row and columns
        keep number of terms same
        if A is a non-null matrix
                for i = 0 to number of columns in A - 1
                        rowTerms [i] = number of rowTerms in ith column
                        startPos[i] = starting position of column as row in transposed matrix
                end for
                for i = 0 to number of non-zero terms in A - 1
                        j = startPos [A[i].col]
                        startPos [A[i].col] = startPos [A[i].col] + 1
                        B[j].row = A[i].col;
```

```
                B[j].col = A[i].row;
                B[j].value = A[i].value;
                end for
            return B
        end if

function add (sparseterm array A, sparseterm array B) :
        C = sparseterm array of size MAX_TERMS
        if  A[0].row == A[0].row  and  A[0].col == B[0].col
            C[0].row =A[0].row
            C[0].col =A[0].col
            i = 1, j = 1
            while  i <= A[0].value  and   j <= B[0].value
                if  A[i] comes before B[j] in row-major order
                    store <A[i].row, A[i].col, A[i].value> in C
                    i = i + 1
                else if  both A[i] and B[i] occur at the same position
                    store <A[i].row, A[i].col, A[i].value + B[i].value> in C
                    i = i +1
                    j = j + 1
                else
                    store <B[i].row, B[i].col, B[i].value> in C
                    j = j + 1
                end if
            end while
            if some elements are yet to be encountered in either A or B
                store these elements in C
            end if
        else    print (“Matrices incompatible for addition”)
        return C

function mult (sparseterm array A, sparseterm array B) :
        if number of columns in A != number of columns in B
            print (“Matrices incompatible for multiplication”)
            return
        end if
        C = sparseterm array of size MAX_TERMS
```

```
newB = sparseterm array of size MAX_TERMS
newB = transpose (B)

A [A[0].value + 1].row = A[0].row
newB [B[0].value + 1].col = B[0].col

totalC = 0, rowBegin = 1, row = A[1].row, column, sum = 0
i = 1, j = 1
while i <= A[0].value
        column = newB[1].row
        while j <= B[0].value + 1
                if no more non-zero term left in previous row of A
                        store <row, column, sum> in C
                        sum = 0
                        i = rowBegin
                        column = next column of B
                else if no more non-zero term left in previous column of B
                        store <row, column, sum> in C
                        sum = 0
                        i = rowBegin
                        column = current column of B
                else
                        if A[i].col < newB[j].col
                                i = i + 1
                        else if A[i].col == newB[j].col
                                sum = sum +  (A[i].value * newB[j].value)
                                i = i + 1, j = j + 1
                        else
                                j = j + 1
                        end if
                end if
        end while
        go to next row in A
        rowBegin = i
        row = A[i].row
end while
C[0].row = A[0].row
```

C[0].col = B[0].col
C[0].value = totalC
return C

RESULTS

```
neeladripal@Neeladris-Macbook-Air Assignment-Set-II-Solutions % gcc -o a2 sol2.c
neeladripal@Neeladris-Macbook-Air Assignment-Set-II-Solutions % ./a2
Sparse Matrix Operations-->
1. Transpose a matrix 2. Add two matrices 3. Multiply two matrices
Enter your choice: 1
Enter a sparse matrix -->
Enter number of rows: 3
Enter number of colmuns: 2
Enter the number of non zero terms: 3
Enter the values in row-major order in <row_no, col_no, value> format -->
0 0 1
1 0 5
1 1 4
Resultant Sparse Matrix -->
Number of rows: 2      Number of columns: 3
Row    Column Value
     0      0      1
     0      1      5
     1      1      4
Do you want to continue (y/n) ?y
Sparse Matrix Operations-->
1. Transpose a matrix 2. Add two matrices 3. Multiply two matrices
Enter your choice: 2
Enter a sparse matrix -->
Enter number of rows: 3
Enter number of colmuns: 4
Enter the number of non zero terms: 4
Enter the values in row-major order in <row_no, col_no, value> format -->
0 1 2
1 3 9
2 0 6
2 2 3
Enter second sparse matrix -->
Enter number of rows: 3
Enter number of colmuns: 4
Enter the number of non zero terms: 5
Enter the values in row-major order in <row_no, col_no, value> format -->
0 0 5
0 1 2
1 3 11
2 2 8
2 3 6
```

```
Resultant Sparse Matrix -->
Number of rows: 3      Number of columns: 4
Row    Column Value
     0      0      5
     0      1      4
     1      3     20
     2      0      6
     2      2     11
     2      3      6
Do you want to continue (y/n) ?y
Sparse Matrix Operations-->
1. Transpose a matrix 2. Add two matrices 3. Multiply two matrices
Enter your choice: 3
Enter a sparse matrix -->
Enter number of rows: 3
Enter number of colmuns: 2
Enter the number of non zero terms: 2
Enter the values in row-major order in <row_no, col_no, value> format -->
0 0 2
1 1 4
Enter second sparse matrix -->
Enter number of rows: 2
Enter number of colmuns: 4
Enter the number of non zero terms: 3
Enter the values in row-major order in <row_no, col_no, value> format -->
0 1 7
0 2 4
1 3 3
Resultant Sparse Matrix -->
Number of rows: 3      Number of columns: 4
Row    Column Value
     0      1     14
     0      2      8
     1      3     12
Do you want to continue (y/n) ?n
neeladripal@Neeladris-Macbook-Air Assignment-Set-II-Solutions %
```

DISCUSSION

Using sparse matrix representation is very memory efficient compared to the 2D array representation which takes O(rows*columns) space, especially when the number of non-zero terms is very less. We store the sparse matrix in row major order.

Functions →

1. transpose (*SparseMatrix* A) - Instead of the general brute force approach, we use

the fast Transpose strategy which initially counts the number of row terms present in the original matrix and arranges them in the new matrix space according to their starting position calculated. Time complexity is O(terms Of Sparse matrix) and space complexity is O(terms of Sparse Matrix).

2. add (*SparseMatrix* A, *SparseMatrix* B) - We compare the terms in A and B to check whose term comes earlier in order and we store the appropriate element in C. This operation takes O(max (no of terms in A, no of terms in B)). And then we store the elements remaining in either A or B in C. Time complexity is O (*number of non-zero terms in A + number of non-zero terms in B*) and the space complexity is same as that of the fast transpose.

3. mult (*SparseMatrix* A, *SparseMatrix* B) - Time complexity is O(*number of columns in B * number of non-zero terms in A + number of columns in A* number of non-zero terms in B*) and the space complexity is the same.

SOURCE CODE

sparse_matrix.h, sol2.c

PROBLEM NUMBER - 3

PROBLEM STATEMENT

Define an ADT for List.

Write C data structure representation and functions for the operations on the List 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. Two data structures with and without using sentinels in arrays are to be implemented.

SOLUTION APPROACH

ADT List

objects: An ordered collection of items of some element type E. The objects are not in sorted order, it just means that each object has a position in the List, starting with position zero.

functions: for all $L \in$ List, $i, j \in$ index, $val \in$ element

1. InitList (*L*) ::= set the first element of L as the end, i.e. length of the list as 0
2. *Integer* LengthList (*L*) ::= **return** the number of elements present in the list.
3. *Integer* GetIth (*L*, *i*) ::= **if** *i* is a valid index **return** *i*th element **else return** error
4. DisplayList (*L*) ::= **print** the list in original order or reverse order, depending on the choice of user.
5. UpdateIth (*L*, *i*, *val*) ::= **if** *i* is a valid index **set** *i*th element = val **else return** error
6. InsertIth (*L*, *i*, *val*) ::= **if** *i* is a valid index and there is space to insert more elements **insert** *val* in position *i* **else return** error
7. DeleteIth (*L*, *i*) ::= **if** *i* is a valid index **remove** element in position *i* **else return** error
8. *Boolean* Search (*L*, *val*) ::= **if** *val* is in list **return** TRUE **else return** FALSE

Representation We represent a list using arrays where each element is an item of the list. We can keep track where the list ends using sentinels or without using sentinels. During input, 1-based indexing is followed.

STRUCTURED PSEUDOCODE

For List using Array with Sentinel →

```
MAX_LEN = 100
SENTINEL = -2*109

structure List :
        elements = array of size MAX_LEN

function initList (List L):
        L.elements [0] = SENTINEL

function lengthList (List L):
        len = 0
        while len < MAX_LEN and L.elements[len] != SENTINEL
                len = len + 1
        end while
        return len

function getIth (List L, i):
        len = lengthList (L)
        i = i – 1
        if i >= 0 and i < len
                return L.elements [i]
        else
                print Out of Bounds Error
                return SENTINEL
        end if

function displayList (List L):
        len = lengthList (L)
```

```
        input choice
        if choice is original order
                for i = 0 to len – 1 in step of 1
                        print L.elements[i]
                end for
        else
                for i = len – 1 to 0 in steps of -1
                        print L.elements[i]
                end for
        end if

function updateIth (List L, i, val):
        len = lengthList (L)
        i = i – 1
        if i >= 0 and i < len
                L.elements [i] = val
        else
                print Out of Bounds Error
        end if

function insertIth (List L, i, val):
        len = lengthList (L)
        if len == MAX_LEN – 1
                print Overflow Error
        else if i >= 1 and i <= len + 1
                i = i – 1
                shift elements towards right by an index, starting from ith index
                L.elements [i] = val
                L.elements [len + 1] = SENTINEL
        else
                print Out of Bounds Error
        end if

function deleteIth (List L, i):
        len = lengthList(L)
        if i >= 1 and i <= len
                i = i – 1
```

```
            shift elements towards left by an index, starting from ith index
        else
            print Out of Bounds Error
        end if

function search (List L, val):
        len = lengthList(L)
        for i = 0 to len – 1 in steps of 1
            if L.elements[i] == val
                return TRUE
            end if
        end for
        return FALSE
```

For List using Array without Sentinel →

```
MAX_LEN = 100

structure List :
        length
        elements [MAX_LEN]

function initList (List L):
        L.length = 0

function lengthList (List L):
        return L.length

function getIth (List L, i):
        len = lengthList (L)
        i = i – 1
        if i >= 0 and i < len
            return L.elements [i]
        else
            print Out of Bounds Error
        end if
```

```
function displayList (List L):
        len = lengthList (L)
        input choice
        if choice is original order
                for i = 0 to len – 1 in step of 1
                        print L.elements[i]
                end for
        else
                for i = len – 1 to 0 in steps of -1
                        print L.elements[i]
                end for
        end if

function updateIth (List L, i, val):
        len = lengthList (L)
        i = i – 1
        if i >= 0 and i < len
                L.elements [i] = val
        else
                print Out of Bounds Error
        end if

function insertIth (List L, i, val):
        len = lengthList (L)
        if len == MAX_LEN – 1
                print Overflow Error
        else if i >= 1 and i <= len + 1
                i = i – 1
                shift elements towards right by an index, starting from ith index
                L.elements [i] = val
                L.elements [len + 1] = SENTINEL
        else
                print Out of Bounds Error
        end if

function deleteIth (List L, i):
        len = lengthList(L)
```

```
        if i >= 1 and i <= len
                i = i – 1
                shift elements towards left by an index, starting from ith index
        else
                print Out of Bounds Error
        end if

function search (List L, val):
        len = lengthList(L)
        for i = 0 to len – 1 in steps of 1
                if L.elements[i] == val
                        return TRUE
                end if
        end for
        return FALSE
```

RESULTS

```
neeladripal@Neeladris-Macbook-Air Assignment-Set-II-Solutions % gcc -o a3 sol3.c
neeladripal@Neeladris-Macbook-Air Assignment-Set-II-Solutions % ./a3

Select what kind of list you want to use 1. Sentinel Version 2. Without Sentinel Version : 1

List Operations (1-based indexing) -->
1. Insert an element in the list.
2. Get the length of the list.
3. Get ith element of the list.
4. Display the list.
5. Update an element of the list.
6. Delete ith element of the list.
7. Search for an element in the list.
Enter your choice: 1

Enter position of insertion: 1

Enter the value to be inserted: 5

Do you want to continue (y/n)? y

List Operations (1-based indexing) -->
1. Insert an element in the list.
2. Get the length of the list.
3. Get ith element of the list.
4. Display the list.
5. Update an element of the list.
```

```
6. Delete ith element of the list.
7. Search for an element in the list.
Enter your choice: 1

Enter position of insertion: 2

Enter the value to be inserted: 3

Do you want to continue (y/n)? y

List Operations (1-based indexing) -->
1. Insert an element in the list.
2. Get the length of the list.
3. Get ith element of the list.
4. Display the list.
5. Update an element of the list.
6. Delete ith element of the list.
7. Search for an element in the list.
Enter your choice: 1

Enter position of insertion: 77

Enter the value to be inserted: 6
Out of bounds. Enter a proper index for insertion.

Do you want to continue (y/n)? y

List Operations (1-based indexing) -->
1. Insert an element in the list.
2. Get the length of the list.
3. Get ith element of the list.
4. Display the list.
5. Update an element of the list.
6. Delete ith element of the list.
7. Search for an element in the list.
Enter your choice: 1

Enter position of insertion: 3

Enter the value to be inserted: 45

Do you want to continue (y/n)? y

List Operations (1-based indexing) -->
1. Insert an element in the list.
2. Get the length of the list.
3. Get ith element of the list.
4. Display the list.
5. Update an element of the list.
6. Delete ith element of the list.
7. Search for an element in the list.
Enter your choice: 2

Length of list: 3
Do you want to continue (y/n)? y
```

```
List Operations (1-based indexing) -->
1. Insert an element in the list.
2. Get the length of the list.
3. Get ith element of the list.
4. Display the list.
5. Update an element of the list.
6. Delete ith element of the list.
7. Search for an element in the list.
Enter your choice: 3

Enter index of element whose value you want to get: 2

Value at index 2: 3
Do you want to continue (y/n)? y

List Operations (1-based indexing) -->
1. Insert an element in the list.
2. Get the length of the list.
3. Get ith element of the list.
4. Display the list.
5. Update an element of the list.
6. Delete ith element of the list.
7. Search for an element in the list.
Enter your choice: 4
How would you prefer to view the list? 1.Original Order 2. Reverse Order
1
List in original order: 5 3 45

Do you want to continue (y/n)? y

List Operations (1-based indexing) -->
1. Insert an element in the list.
2. Get the length of the list.
3. Get ith element of the list.
4. Display the list.
5. Update an element of the list.
6. Delete ith element of the list.
7. Search for an element in the list.
Enter your choice: 5

Enter index of element to be updated: 2

Enter the new value: 58

Do you want to continue (y/n)? y

List Operations (1-based indexing) -->
1. Insert an element in the list.
2. Get the length of the list.
3. Get ith element of the list.
4. Display the list.
5. Update an element of the list.
6. Delete ith element of the list.
7. Search for an element in the list.
Enter your choice: 4
```

```
How would you prefer to view the list? 1.Original Order 2. Reverse Order
2
List in reverse order: 45 58 5

Do you want to continue (y/n)? y

List Operations (1-based indexing) -->
1. Insert an element in the list.
2. Get the length of the list.
3. Get ith element of the list.
4. Display the list.
5. Update an element of the list.
6. Delete ith element of the list.
7. Search for an element in the list.
Enter your choice: 6

Enter index of element to be deleted: 3

Do you want to continue (y/n)? y

List Operations (1-based indexing) -->
1. Insert an element in the list.
2. Get the length of the list.
3. Get ith element of the list.
4. Display the list.
5. Update an element of the list.
6. Delete ith element of the list.
7. Search for an element in the list.
Enter your choice: 7

Enter value of element to be searched: 45

Element not found.
Do you want to continue (y/n)? y

List Operations (1-based indexing) -->
1. Insert an element in the list.
2. Get the length of the list.
3. Get ith element of the list.
4. Display the list.
5. Update an element of the list.
6. Delete ith element of the list.
7. Search for an element in the list.
Enter your choice: 4
How would you prefer to view the list? 1.Original Order 2. Reverse Order
1
List in original order: 5 58

Do you want to continue (y/n)? y

List Operations (1-based indexing) -->
1. Insert an element in the list.
2. Get the length of the list.
```

```
3. Get ith element of the list.
4. Display the list.
5. Update an element of the list.
6. Delete ith element of the list.
7. Search for an element in the list.
Enter your choice: 6

Enter index of element to be deleted: 1

Do you want to continue (y/n)? y

List Operations (1-based indexing) -->
1. Insert an element in the list.
2. Get the length of the list.
3. Get ith element of the list.
4. Display the list.
5. Update an element of the list.
6. Delete ith element of the list.
7. Search for an element in the list.
Enter your choice: 4
How would you prefer to view the list? 1.Original Order 2. Reverse Order
1
List in original order: 58

Do you want to continue (y/n)? n
neeladripal@Neeladris-Macbook-Air Assignment-Set-II-Solutions % $
```

DISCUSSION

Sentinel is a value which marks the end of the list. When not using sentinels, we keep a length property in the List itself to keep track of the number of elements in the list. We accept 1-based indices in the functions and convert them to 0-based indices within the function for working with values.

Functions →

1. initList (*List* L) - This initializes the list and sets the first element as the end of the list; works in O (1) time complexity.
2. lengthList (*List* L) - The complexity for the List ADT without sentinel will be O(1), as the structure contains a member keeping track of the number of elements in the list. On the other hand, for List ADT with sentinel, the complexity will be O(n), as it will traverse the whole list till it finds the sentinel.
3. getIth (*List* L, i) -The complexity for the List ADT without sentinel will be O(1), whereas for List ADT with sentinel it'll be O(n), as finding length takes O(n) time.

4. diaplayList (*List* L) -This function will traverse through the whole list and print it, hence the complexity is O(n).
5. updateIth (*List* L, i, val) -The complexity for the List ADT without sentinel will be O(1), whereas for List ADT with sentinel it'll be O(n), as finding length takes O(n) time.
6. insertIth (*List* L, i, val) - This function will update all the positions from i to the end, hence in the worst case, it'll have to traverse all the elements. Hence, the complexity will be O(n).
7. deleteIth (*List* L, int i) - This function will update all the positions from i to the end, hence in the worst case, it'll have to traverse all the elements. Hence, the complexity will be O(n) - The process is very similar to the insertIth function.
8. search (*List* L, val) - Since, the List ADT is not sorted, hence we have to go for Linear search, which takes O(n) time.

SOURCE CODE

listWithSentinel.h, listWithoutSentinel.h, sol3.c

PROBLEM NUMBER - 4

PROBLEM STATEMENT

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

ADT Set

objects: An unordered collection of items which have unique value.

functions: for all $S \in$ Set, $i, j \in$ index, *item* $\in$ element

1. InitSet (*S*) ::= set size of *S* = 0
2. *Bool* FindItem (*S, item*) ::= **if** *item* $\in$ *S* **return** TRUE **else return** FALSE
3. InsertItem (*S, item*) ::= **if** *item* $\in$ *S* **return** error **else** insert *key* in *S*
4. RemoveItem (*S, item*) ::= **if** *item* $\in$ *S* remove *item* from *S* **else return** error
5. Display (*S*) ::= **print** contents of *S*
6. *Set* UnionOf (*S1, S2*) ::= **return** a set formed as *S1* $\cup$ *S2*
7. *Set* IntersectionOf (*S1, S2*) ::= **return** a set formed as *S1* $\cap$ *S2*
8. *Set* DifferenceOf (*S1, S2*) ::= **return** a set formed as *S1* - *S2*
9. *Bool* IsSubset (*S1, S2*) ::= **if** *S1* is a subset of *S2* **return** TRUE **else return** FALSE

Representation We represent a set using a structure which contains a property *size* (denoting size of set) and an array of elements which contains the items of the set. No two items of a set (here, elements of the array) can have the same value.

STRUCTURED PSEUDOCODE

MAX = some large natural number

```
structure Set:
        size
        data = array of size MAX

function initSet (Set S) :
        S.size = 0

function findItem (Set S, item):
        for i = 0 to S.size – 1 in steps of 1
                if S.data[i] == item
                        return TRUE
                end if
        end for
        return FALSE

function insertItem (Set S, item):
        if item is not in S
                if size < MAX
                        S.data [size] = item
                        S.size = s.size + 1
                end if
        else
                print Error
        end if

function removeItem (Set S, item):
        if S.size == 0
                print Empty Set
        else if item is not in S
                print item not present in S
        else
                i = index of the item to be removed
                shift items left by one position, starting from index i
                S.size = S.size - 1
        end if

function display (Set S):
```

```
        for i = 0 to S.size – 1 in steps of 1
                print S.data [i]
        end for

function unionOf (Set S1, Set S2):
        init Set S
        for i = 0 to S1.size – 1 in steps of 1
                insertItem (S, S1.data [i])
        end for
        for i = 0 to S2.size – 1 in steps of 1
                insertItem (S, S2.data[i])
        end for
        return S

function intersectionOf (Set S1, Set S2):
        init Set S
        for j = 0 to S1.size – 1 in steps of 1
                if S1.data[j] is in S2
                        insertItem (S, S1.data[j])
                end if
        end for
        return S

function differenceOf (Set S1, Set S2):
        init Set S
        for i = 0 to S1.size – 1 in steps of 1
                if S1.data[i] is not in S2
                        insertItem (S, S1.data[i])
                end if
        end for
        return S

function isSubSet (Set S1, Set S2):
        for i = 0 to S1.size – 1 in steps of 1
                if S1.data[i] is not in S2
                        return FALSE
                end if
```

end for
return TRUE

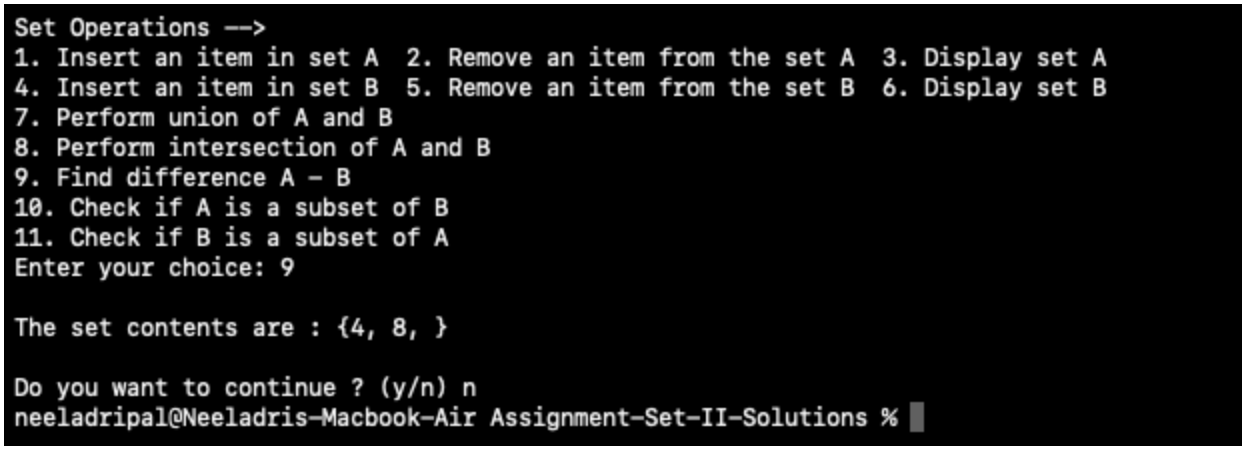
RESULTS

```
neeladripal@Neeladris-Macbook-Air Assignment-Set-II-Solutions % gcc -o a4 sol4.c
neeladripal@Neeladris-Macbook-Air Assignment-Set-II-Solutions % ./a4

Set Operations -->
1. Insert an item in set A  2. Remove an item from the set A  3. Display set A
4. Insert an item in set B  5. Remove an item from the set B  6. Display set B
7. Perform union of A and B
8. Perform intersection of A and B
9. Find difference A - B
10. Check if A is a subset of B
11. Check if B is a subset of A
Enter your choice: 1

Enter item value: 3

Do you want to continue ? (y/n) y

Set Operations -->
1. Insert an item in set A  2. Remove an item from the set A  3. Display set A
4. Insert an item in set B  5. Remove an item from the set B  6. Display set B
7. Perform union of A and B
8. Perform intersection of A and B
9. Find difference A - B
10. Check if A is a subset of B
11. Check if B is a subset of A
Enter your choice: 1

Enter item value: 1

Do you want to continue ? (y/n) y

Set Operations -->
1. Insert an item in set A  2. Remove an item from the set A  3. Display set A
4. Insert an item in set B  5. Remove an item from the set B  6. Display set B
7. Perform union of A and B
8. Perform intersection of A and B
9. Find difference A - B
10. Check if A is a subset of B
11. Check if B is a subset of A
Enter your choice: 1

Enter item value: 4

Do you want to continue ? (y/n) y
```

```
Set Operations -->
1. Insert an item in set A  2. Remove an item from the set A  3. Display set A
4. Insert an item in set B  5. Remove an item from the set B  6. Display set B
7. Perform union of A and B
8. Perform intersection of A and B
9. Find difference A - B
10. Check if A is a subset of B
11. Check if B is a subset of A
Enter your choice: 4

Enter item value: 3

Do you want to continue ? (y/n) y

Set Operations -->
1. Insert an item in set A  2. Remove an item from the set A  3. Display set A
4. Insert an item in set B  5. Remove an item from the set B  6. Display set B
7. Perform union of A and B
8. Perform intersection of A and B
9. Find difference A - B
10. Check if A is a subset of B
11. Check if B is a subset of A
Enter your choice: 4

Enter item value: 1

Do you want to continue ? (y/n) y

Set Operations -->
1. Insert an item in set A  2. Remove an item from the set A  3. Display set A
4. Insert an item in set B  5. Remove an item from the set B  6. Display set B
7. Perform union of A and B
8. Perform intersection of A and B
9. Find difference A - B
10. Check if A is a subset of B
11. Check if B is a subset of A
Enter your choice: 3

The set contents are : {3, 1, 4, }

Do you want to continue ? (y/n) y

Set Operations -->
1. Insert an item in set A  2. Remove an item from the set A  3. Display set A
4. Insert an item in set B  5. Remove an item from the set B  6. Display set B
7. Perform union of A and B
8. Perform intersection of A and B
9. Find difference A - B
10. Check if A is a subset of B
11. Check if B is a subset of A
Enter your choice: 6

The set contents are : {3, 1, }

Do you want to continue ? (y/n) y
```

```
Set Operations -->
1. Insert an item in set A  2. Remove an item from the set A  3. Display set A
4. Insert an item in set B  5. Remove an item from the set B  6. Display set B
7. Perform union of A and B
8. Perform intersection of A and B
9. Find difference A - B
10. Check if A is a subset of B
11. Check if B is a subset of A
Enter your choice: 10

A is not a subset of B.
Do you want to continue ? (y/n) y

Set Operations -->
1. Insert an item in set A  2. Remove an item from the set A  3. Display set A
4. Insert an item in set B  5. Remove an item from the set B  6. Display set B
7. Perform union of A and B
8. Perform intersection of A and B
9. Find difference A - B
10. Check if A is a subset of B
11. Check if B is a subset of A
Enter your choice: 11

B is a subset of A.
Do you want to continue ? (y/n) y

Set Operations -->
1. Insert an item in set A  2. Remove an item from the set A  3. Display set A
4. Insert an item in set B  5. Remove an item from the set B  6. Display set B
7. Perform union of A and B
8. Perform intersection of A and B
9. Find difference A - B
10. Check if A is a subset of B
11. Check if B is a subset of A
Enter your choice: 1

Enter item value: 8

Do you want to continue ? (y/n) y

Set Operations -->
1. Insert an item in set A  2. Remove an item from the set A  3. Display set A
4. Insert an item in set B  5. Remove an item from the set B  6. Display set B
7. Perform union of A and B
8. Perform intersection of A and B
9. Find difference A - B
10. Check if A is a subset of B
11. Check if B is a subset of A
Enter your choice: 4

Enter item value: 9

Do you want to continue ? (y/n) y
```

```
Set Operations -->
1. Insert an item in set A  2. Remove an item from the set A  3. Display set A
4. Insert an item in set B  5. Remove an item from the set B  6. Display set B
7. Perform union of A and B
8. Perform intersection of A and B
9. Find difference A - B
10. Check if A is a subset of B
11. Check if B is a subset of A
Enter your choice: 3

The set contents are : {3, 1, 4, 8, }

Do you want to continue ? (y/n) y

Set Operations -->
1. Insert an item in set A  2. Remove an item from the set A  3. Display set A
4. Insert an item in set B  5. Remove an item from the set B  6. Display set B
7. Perform union of A and B
8. Perform intersection of A and B
9. Find difference A - B
10. Check if A is a subset of B
11. Check if B is a subset of A
Enter your choice: 6

The set contents are : {3, 1, 9, }

Do you want to continue ? (y/n) y

Set Operations -->
1. Insert an item in set A  2. Remove an item from the set A  3. Display set A
4. Insert an item in set B  5. Remove an item from the set B  6. Display set B
7. Perform union of A and B
8. Perform intersection of A and B
9. Find difference A - B
10. Check if A is a subset of B
11. Check if B is a subset of A
Enter your choice: 7

The set contents are : {3, 1, 4, 8, 9, }

Do you want to continue ? (y/n) y

Set Operations -->
1. Insert an item in set A  2. Remove an item from the set A  3. Display set A
4. Insert an item in set B  5. Remove an item from the set B  6. Display set B
7. Perform union of A and B
8. Perform intersection of A and B
9. Find difference A - B
10. Check if A is a subset of B
11. Check if B is a subset of A
Enter your choice: 8

The set contents are : {3, 1, }

Do you want to continue ? (y/n) y
```



DISCUSSION

The items of a set need to have a unique value. So, every time we perform an operation which modifies the set, we need to check if the item belongs to the set.

Functions →

1. initSet (*Set* S) - This sets size = 0; works in O(1) time complexity.
2. findItem (*Set* S, item) - This function traverses the whole list to check for the presence of the item in the set; works in O(n) time complexity.
3. insertItem (*Set* S, item) - This function traverses the whole set, checks if the item is present already in the set, and if not present adds the item at the end of the set and increases size of the set by 1; works in O(n) time complexity.
4. removeItem (*Set* S, item) - This function checks if the item is present in the set or not. If present, then traverses till the position of occurrence and deletes the node by shifting the remaining items to the last left by one index and reduces the size of the set by 1; works in O(n) time complexity.
5. display (*Set* S) - This function traverses through the whole set printing the elements; works in O(n) time complexity.
6. unionOf (*Set* S1, *Set* S2) - This function inserts all items of both sets into the union set, and returns the union set. The traversal part takes O(n) time and insertion also takes O(n) time, hence, the overall time complexity becomes $O(n^2)$.

7. intersectionOf (*Set* S1, *Set* S2) - This function inserts all items which are common to both sets, and returns the intersection set. The traversal part of one set takes O(n) time, and checking its presence in the other set takes O(n) time as well as the insertion takes O(n) time, hence overall time complexity is $O(n^2)$.

8. differenceOf (*Set* S1, *Set* S2) - This function inserts all items which are present in the first set but not present in the second set. The traversal part takes O(n) time, and checking its presence in the other set takes O(n) time as well as the insertion takes O(n) time, hence overall time complexity is $O(n^2)$.

9. isSubset (*Set* S1, *Set* S2) - This function checks if the first set is a subset of the second set. The traversal of the first set takes O(n) time, and checking its presence in the second set takes O(n) time, hence overall time complexity is $O(n^2)$.

SOURCE CODE

set.h, sol4.c

PROBLEM NUMBER - 5

PROBLEM STATEMENT

Write C data representation and functions for the operations on the String in a Header file, with array as the base data structure, without using any inbuilt function in C.

Write a menu-driven main program in a separate file for testing the different operations and include the above header file.

SOLUTION APPROACH

ADT String

objects: A finite set of zero or more characters, terminated by a delimiter character, usually the NULL character.

functions: for all *S, S1, S2* ∈ String, *m* ∈ Natural number, *item* ∈ element, *i* ∈ index

1. *String* null (*m*) ::= **return** an Empty string whose capacity = m characters
2. *String* input () ::= **return** a String whose characters are taken input
3. Display (S) ::= **print** the characters of S
4. *Integer* Length (*S*) ::= **returns** number of characters in S
5. *Integer* Compare (*S1, S2*) ::= compare *S1* and *S2* lexicographically, **return** 0 if *S1* == *S2*, **return** 1 if *S1* > *S2*, **return** -1 if *S1* < *S2*
6. *Bool* IsNull (*S)* ::= **if** *S* has 0 characters **return** TRUE **else return** FALSE
7. *String* Concat (*S1, S2*) ::= **return** a string formed by concatenating *S2* to *S1*
8. *String* Substr (*S, i, m*) ::= **return** substring of *S* of length *m* starting at index *i*
9. Clear (S) ::= delete the String from memory

Representation We implement String using dynamically allocated character array. Characters are stored in this array. End of string is marked by '\0' (NULL) character.

STRUCTURED PSEUDOCODE

structure String:

s = dynamically allocated character array

```
function null (m):
        init String S
        dynamically allocate (m+1) space in S
        S.s[0] = '\0'
        return S

function input () :
        input str = a stream of characters
        len = 0
        while str[len] != '\0'
                len = len + 1
        end while
        S = null (len)
        for i = 0 to len in steps of 1
                S.s[i] = str[i]
        end for
        return S

function output (String S):
        print S.s

function length (String S):
        len = 0
        while S.s[len] != '\0'
                len = len + 1
        end while
        return len

function compare (String S1, String S2):
        len1 = length (S1), len2 = length (S2)
        i = 0

        while i < len1 and i < len2
                if S1.s[i] == S2.s[i]
                        i = i + 1
                        continue to next iteration
                else if S1.s[i] < S2.s[i]
```

```
                return -1
        else
                return 1
        end if
    end while

    if both S1 and S2 has come to their end
        return 0
    else if first string has a larger length
        return 1
    else
        return -1
    end if

function isNull (String S):
    len = length (S)
    nullS = null (len)
    if compare (S, nullS) == 0
        return true
    else
        return false
    end if

function concat (String S1, String S2) :
    len1 = length (S1), len2 = length (S2)
    S = null (lena + lenb)
    copy characters of S1 in S
    copy characters of S2 in S
    set the value of last element of S.s to '\0'
    return S

function substr (String S, i, m):
    len = length (S)
    if substring can be formed with starting index at i and length m
        sub = null string with capacity 'm'
        copy characters of substring into sub
        set the value of last element of sub.s to '\0'
```

```
        return sub
    else
        return Empty String
    end if

function clear (String S) :
    free S.s
```

RESULTS

```
[neeladripal@Neeladris-Macbook-Air Assignment-Set-II-Solutions % gcc -o a5 sol5.c
[neeladripal@Neeladris-Macbook-Air Assignment-Set-II-Solutions % ./a5

String Operations -->
1. Display length of a string.
2. Compare two strings lexicographically.
3. Concatenate two strings.
4. Return a substring of a string.
Enter your choice: 1

Enter the string: it is 7 in the morning

Length of the entered string is 22.
Do you want to continue (y/n) ? y

String Operations -->
1. Display length of a string.
2. Compare two strings lexicographically.
3. Concatenate two strings.
4. Return a substring of a string.
Enter your choice: 2

Enter the first string: dsa assignment

Enter the second string: sleep

Second string is lexicographically greater.
Do you want to continue (y/n) ? y
```

```
String Operations -->
1. Display length of a string.
2. Compare two strings lexicographically.
3. Concatenate two strings.
4. Return a substring of a string.
Enter your choice: 3

Enter the first string: i need to t

Enter the second string: ake a nap

Concatenated string: i need to take a nap

Do you want to continue (y/n) ? y

String Operations -->
1. Display length of a string.
2. Compare two strings lexicographically.
3. Concatenate two strings.
4. Return a substring of a string.
Enter your choice: 4

Enter the string: it consumed 3 whole days

Length of the entered string is 24.
Enter starting index of substring (0-based): 8

Enter length of the substring: 8

Substring Requested: med 3 wh

Do you want to continue (y/n) ? n
neeladripal@Neeladris-Macbook-Air Assignment-Set-II-Solutions %
```

DISCUSSION

The structure String has a character pointer which points to the dynamically allocated array of characters. This array stores the content of the string.

Functions →

1. null (m) - This function dynamically allocates (m + 1) characters in the memory for the string.
2. input () - This function takes the input of the string and copies it to a String variable. A whole traversal takes place during copying, hence, the time complexity is O(n).

3. display (*String* S) - This function prints the string, and the time complexity of printing a string of n characters is O(n).

4. length (*String* S) - This function traverses through the string, until it finds the delimiter character, i.e. '\0', hence, the time complexity is O(n), where n is the length of the string.

5. compare (*String* S1, *String* S2) - This function traverses through both the strings until one of the strings comes to its end, or a mismatch is found. So, in the worst case it will end up traversing the whole string which has a smaller length. Hence, this method also takes O(n) time.

6. isNull (*String* S) - This function generates a null string using the null method and also calls the compare method to check if this is a null string. Now, in the compare function, one String is always a NULL (or, empty) string, hence, it won't traverse any index. So, the time complexity should be O(1).

7. concat (*String* S1, *String* S2) - This function traverses both strings and concats the second one to the first one. Hence, the time complexity will be O (len1 + len2), where len1 and len2 are the string lengths of the Strings S1 and S2.

8. substr (*String* S, i, m) - This function checks if the substring requested is a valid one. If valid, it traverses from the starting point for 'm' length and copies the characters to a new String sub, which the function returns. The loop traverses 'm' times, and in the worst case, 'm' can be the whole string, hence, the time complexity will be O(n).

9. clear (*String* S) - This function frees the dynamically allocated memory for the string.

SOURCE CODE

stringadt.h, sol5.c

PROBLEM NUMBER - 6

PROBLEM STATEMENT

Given a large single dimensional array of integers, write functions for sliding window filters with maximum, minimum, median, and average to generate an output array. The window size should be an odd integer like 3, 5 or 7. Explain what you will do with the boundary values.

SOLUTION APPROACH

We are creating utility functions which would return minimum, maximum and median of an array passed as a parameter. Another utility function is created to sort an array. Now, for sliding window filter (of size, say k), we start from 0th index to (n-1)th index and, for each index i, the current window will be in the range (i – k/2) to (i + k/2), consisting of 2*(k-1)/2 + 1 = k elements. Now, if any index lies within, 0 to (n-1), then the value is taken from the array, else, 0 is taken. This is how the boundary values are handled.

STRUCTURED PSEUDOCODE

```
function minarr (arr, n):
        res = arr[0]
        for i = 0 to n – 1 in steps of 1
                if arr[i] < res
                        res = arr[i]
                end if
        end for
        return res

function maxarr (arr, n):
        res = arr[0]
        for i = 0 to n – 1 in steps of 1
                if arr[i] > res
                        res = arr[i]
                end if
```

```
        end for
        return res

function sort (arr, n) :
        for i = 0 to n – 1 in steps of 1
                for j = i + 1 to n – 1 in steps of 1
                        if arr[i] > arr[j]
                                swap arr[i] and arr[j]
                        end if
                end for
        end for

function medianarr (arr, n) :
        sort arr
        return arr[n / 2]

function main () :
        input n = no of elements, arr = array of elements, k = sliding window size
        if k is odd
                minfilter = array of size n
                maxfilter = array of size n
                medianfilter = array of size n
                for i = 0 to n – 1 in steps of 1
                        filter = array of size k
                        x = 0
                        for j = i – k/2 to i + k/2 in steps of 1
                                if j is within bounds of arr
                                        filter[x] = arr[j]
                                else
                                        filter[x] = 0
                                end if
                                x = x + 1
                        end for
                        minfilter [i] = minarr (filter, k)
                        maxfilter [i] = maxarr (filter, k)
                        medianfilter [i] = medianarr (filter, k)
                print minfilter, maxfilter, medianfilter
```

```
        else
            exit
```

RESULTS

```
[neeladripal@Neeladris-Macbook-Air Assignment-Set-II-Solutions % gcc -o a6 sol6.c
[neeladripal@Neeladris-Macbook-Air Assignment-Set-II-Solutions % ./a6

Enter no of elements: 12

Enter 12 elements: 4 5 1 13 3 25 27 18 10 3 4 9

Enter sliding window filter size choice (3 / 5 / 7): 3

Max filter output: 5 5 13 13 25 27 27 27 18 10 9 9
Min filter output: 0 1 1 1 3 3 18 10 3 3 3 0
Median filter output: 4 4 5 3 13 25 25 18 10 4 4 4
[neeladripal@Neeladris-Macbook-Air Assignment-Set-II-Solutions % ./a6

Enter no of elements: 6

Enter 6 elements: 3 66 18 24 9 39

Enter sliding window filter size choice (3 / 5 / 7): 5

Max filter output: 66 66 66 66 39 39
Min filter output: 0 0 3 9 0 0
Median filter output: 3 18 18 24 18 9
neeladripal@Neeladris-Macbook-Air Assignment-Set-II-Solutions %
```

DISCUSSION

The utility functions minarr and maxarr both have time complexities of O(n). For the sorting, we are using bubble sort, hence it's time complexity is $O(n^2)$. Since, the utility function medianarr calls the sort function in it, it's time complexity also becomes $O(n^2)$. In the main function we create 3 arrays of size n and 1 array of size k for storing results, hence the auxiliary space complexity becomes O(3n + k) = O(n). Now, for the time complexity of the main function, for minfilter it is O(n*k), for maxfilter it is O(n*k) and for median filter it is $O(n*k^2)$. Hence, the total time complexity is $O(n*k^2)$.

SOURCE CODE

sol6.c

PROBLEM NUMBER - 7

PROBLEM STATEMENT

Take an arbitrary Matrix of positive integers, say, 128 X 128. Also take integer matrices of size 3 X 3 and 5 X 5. Find out an output matrix of size 128 X 128 by multiplying the small matrix with the corresponding submatrix of the large matrix with the centre of the small matrix placed at the individual positions within the large matrix. Explain how you will handle the boundary values.

SOLUTION APPROACH

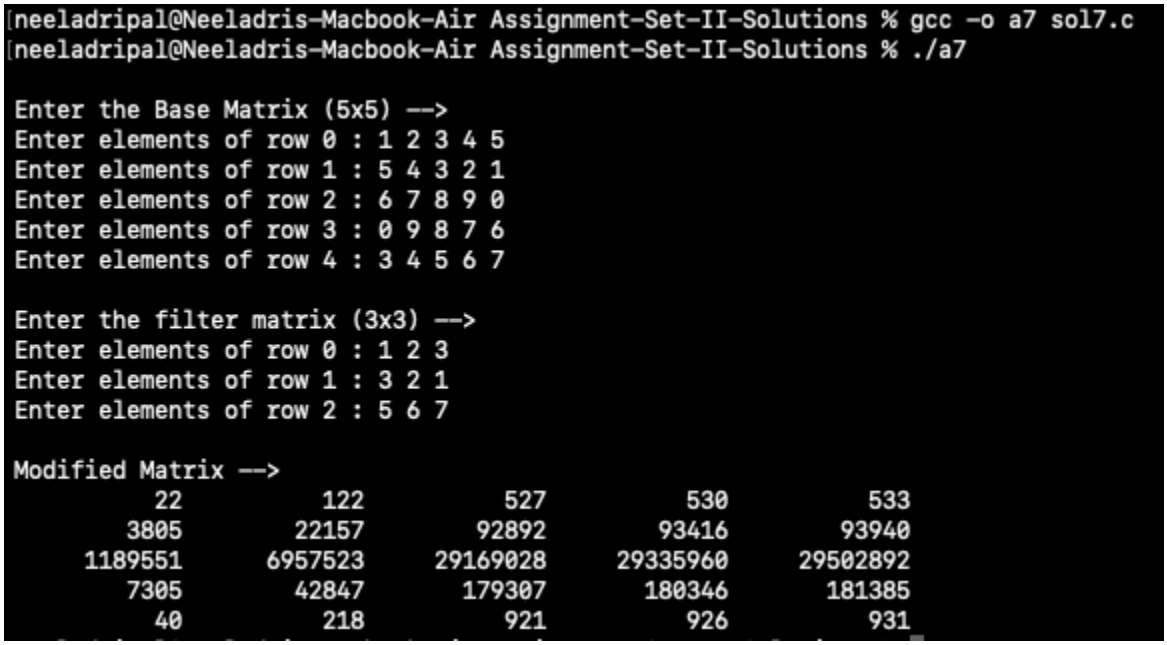
Matrix Multiplication is applied on all possible indices (<cx, cy> where cx is the row number and cy is the column number) of the base matrix (A) with the filter matrix (filter). The boundary condition is that cx & cy must be greater than (or equals) [sizeof (filter) / 2] and less than (sizeof(A) – [sizeof(filter)/2]). (here, [.] denotes the floor function)

STRUCTURED PSEUDOCODE

```
function getMatrix (n) :
        A = matrix of size n * n
        input A
        return A

function applyFilter (A, n, filter, n1, cx, cy) :
        temp = matrix of size n1
        if cx, cy satisfy the boundary conditions
                for i = cx – n/2 to cx + n/2 in steps of 1
                        for j = 0 to n1 in steps of 1
                                sum = 0, k1 = 0
                                for k = cy – n1/2 to cy + n/2 in steps of 1
                                        sum = sum + A[i][k] * B[k1][j]
                                        k1++
                                temp[i – cx + n1/2][j] = sum
                                end for
                        end for
```

```
            end for
        end if

        for i = cx – n1/2 to cx + n1/2 in steps of 1
            for j = cy – n1/2 to cy + n1/2 in steps of 1
                A[i][j] = temp [i – cx + n1/2][j – cy + n1/2]
            end for
        end for

function printMatrix (A, n) :
        for i = 0 to n in steps of 1
            for j = 0 to n in steps of 1
                print A[i][j]
            end for
        end for

function clearMatrix (A, n) :
        free memory occupied by A

function main () :
        n = 5
        A = matrix of size n x n
        input A
        n1 = 3
        filter = matrix of size n1 x n1
        input filter

        cx = row number, cy = column number of centre element of filter matrix
        for cx = 0 to n-1 in steps of 1
            for cy = 0 to n-1 in steps of 1
                applyfilter (A, n, filter, n1, cx, cy)
            end for
        end for

        print A
```

RESULTS



DISCUSSION

The time complexity for the code is $O(n^2 \times n1^3)$ because we need to consider all the indices of the base matrix to apply the filter and to apply the filter we need to multiply the filter matrix with a valid submatrix of the base matrix (A). The space complexity for the code is $O(n^2 + n1^2)$ because we're dealing with two square matrices of size n and n1 respectively.

SOURCE CODE

sol7.c

PROBLEM NUMBER - 8

PROBLEM STATEMENT

Find whether an array is sorted or not, and the sorting order.

SOLUTION APPROACH

An array is taken as input. If the array has 1 or 2 elements, it is a trivial case. If the array has more than 3 elements, for all sets of 3 consecutive elements we check if the 3 elements are in a particular order: either increasing or decreasing. If there exist 3 elements such that they do not maintain a fixed order, the array must be unsorted.

STRUCTURED PSEUDOCODE

```
function isSorted (a) :
        if a has 1 or 2 elements
                return TRUE
        end if

        for i = 2 to n - 1 in steps of 1
                if a[i] > a[i-1] < a[i-2] or a[i] < a[i-1] > a[i-2]
                        return FALSE
                end if
        end for

function main () :
        n = no of elements
        a  = array of size n
        input a, n
        if n is positive
                if isSorted (a) == TRUE
                        print Array is sorted
                else
                        print Array is not sorted
                end if
```

```
else
        print n must be positive
end if
```

RESULTS

```
neeladripal@Neeladris-Macbook-Air Assignment-Set-II-Solutions % gcc -o a8 sol8.c
neeladripal@Neeladris-Macbook-Air Assignment-Set-II-Solutions % ./a8
Enter number of terms: 5
Enter elements of the array: 2 6 37 51 67
Array is sorted.
neeladripal@Neeladris-Macbook-Air Assignment-Set-II-Solutions % ./a8
Enter number of terms: 7
Enter elements of the array: 52 37 37 23 19 17 5
Array is sorted.
neeladripal@Neeladris-Macbook-Air Assignment-Set-II-Solutions % ./a8
Enter number of terms: 8
Enter elements of the array: 3 67 45 20 48 42 88 65
Array is not sorted.
neeladripal@Neeladris-Macbook-Air Assignment-Set-II-Solutions %
```

DISCUSSION

Since we traverse the array in O(n) time and perform the comparison operation in O(1) time, the overall time complexity is O(n). Also, the space complexity is O(1) because no auxiliary data structure except the iterator variable is used for storage purposes.

SOURCE CODE

sol8.c

PROBLEM NUMBER - 9

PROBLEM STATEMENT

Given two sorted arrays, write a function to merge the array in the sorting order.

SOLUTION APPROACH

Two sorted arrays are taken as input. We keep two pointers, one for traversing each array. Now we iterate over both arrays simultaneously in such a manner that an element with a lesser value comes first in the resultant array. If we complete iterating over an array and there are still some elements left in the other array, we append these elements to the resultant array.

STRUCTURED PSEUDOCODE

```
function merge (a, b, c) :
        i = 0, j = 0, k = 0
        na = length of array a
        nb = length of array b

        while i < na and j < nb
                if a[i] < b[i]
                        c[k] = a[i]
                        k = k + 1, i = i + 1
                else
                        c[k] = b[j]
                        k = k + 1, j = j + 1
                end if
        end while

        while i < na
                c[k] = a[i]
                k = k + 1, i = i + 1
        end while
```

```
        while j < nb
                c[k] = b[j]
                k = k + 1, j = j + 1
        end while

function main ()
        input na, nb
        input a = array of size na
        input b = array of size nb
        c = array of size na + nb
        merge (a, b, c)
        print c
```

RESULTS

```
[neeladripal@Neeladris-Macbook-Air Assignment-Set-II-Solutions % gcc -o a9 sol9.c
[neeladripal@Neeladris-Macbook-Air Assignment-Set-II-Solutions % ./a9
Enter first array -->
Enter number of terms: 5
Enter elements in increasing order of value: 3 6 45 87 99
Enter second array -->
Enter number of terms: 8
Enter elements in increasing order of value: 4 7 25 41 49 66 72 75
Merged Array: 3 6 4 7 25 41 49 66 72 75 45 87 99
neeladripal@Neeladris-Macbook-Air Assignment-Set-II-Solutions %
```

DISCUSSION

We traverse both the arrays to merge them. So time complexity is O (sum of lengths of both arrays). The merged array is of size = length of array1 + length of array 2. So, space complexity is also the same.

SOURCE CODE

sol9.c