UNIT I – INTRODUCTION

Data structures - Abstract data types - Primitive data structures- – Performance analysis – Space complexity – Time complexity – Asymptotic notations – Performance measurement – Array as an abstract data type – Polynomial as an abstract data type – Sparse matrix abstract data type – String abstract data type.

SPARSE MATRIX ADT

A matrix is a two-dimensional data object made of m rows and n columns, therefore having total m x n values. If most of the elements of the matrix have 0 value, then it is called a sparse matrix.

Advantage of using Sparse Matrix instead of simple matrix:

Storage: There are lesser non-zero elements than zeros and thus lesser memory can be used to store only those elements.

Computing time: Computing time can be saved by logically designing a data structure traversing only non-zero elements..

0 0 3 0 4
0 0 5 7 0
0 0 0 0 0
0 2 6 0 0

Representing a sparse matrix

Representing a sparse matrix by a 2D array leads to wastage of lots of memory as zeroes in the matrix are of no use in most of the cases. So, instead of storing zeroes with non-zero elements, we only store non-zero elements. This means storing non-zero elements with triples- (Row, Column, value).

2D array is used to represent a sparse matrix in which there are three rows named as

- Row: Index of row, where non-zero element is located
- Column: Index of column, where non-zero element is located
- Value: Value of the non zero element located at index – (row,column)

$$\begin{pmatrix} 0 & 0 & 3 & 0 & 4 \\ 0 & 0 & 5 & 7 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 2 & 6 & 0 & 0 \end{pmatrix} \longrightarrow$$

4	5	6
1	3	3
1	5	4
2	3	5
2	4	7
4	2	2
4	3	6

Sparse Matrix Operations:

Addition

In order to combine the matrices, a user can simply go over each element of each matrix, one at a time, and insert the smaller element—the one with the smaller row and column values—into the resulting matrix. If an element has the same column and row values as another element, we add their values and insert the new data into the final matrix.

Example:

We have to add these two matrices from the table presentation. So, we need one more structure of coordinates that is the same type of matrix ‘a’ and ‘b’ but we don’t know how many elements it might be getting. So, what should be the size? The size of the 3rd structure will be = no. of non-zero elements in (Matrix A + Matrix B) = 6 + 6 = 12.

Transpose

To make our comparisons easier and keep the sorted order, firstly calculate the transpose of the 2nd matrix before multiplying the matrices. In order to obtain the resultant matrix, the length of both matrices must be traversed, and the relevant multiplied values must be added.

Any row in the 1st matrix with a value equal to x and any row in the 2nd matrix (the transposed one) with a value equal to y will contribute to the result[x][y]. The result [x][y] is derived by multiplying all elements with col values in both matrices and only adding those with row values of x in the original matrix and y in the 2nd transposed matrix.

Sparse Matrix Addition Algorithm:

r1=A[0][0]; c1=A[0][1]

r2=B[0][0]; c2=B[0][1]

```
If r1!=r2 or c1!=c2 then
        Print "Incompatible matrices"
Else
{
        C[0][0]=A[0][0]
        C[0][1]=A[0][1]
        m=1; n=1; k=1;
        For i=0 to r-1 do
        {
                For j=0 to c1-1 do
                {
                        If A[m][0] = i and A[m][1] = j and B[n][0] = i and B[n][1] = j then
                        {
                                C[k][0] = A[m][0]
                                C[k][1] = A[m][1]
                                C[k][2] = A[m][2] + B[n][2]
                                m=m+1; n=n+1; k=k+1;
                        }
                        Else if A[m][0] = i and A[m][1] = j then
                        {
                                C[k][0] = A[m][0]
                                C[k][1] = A[m][1]
                                C[k][2] = A[m][2]
                                m=m+1; k=k+1;
                        }
                        Else if B[n][0] = i and B[n][1] = j then
                        {
                                C[k][0] = B[n][0]
                                C[k][1] = B[n][1]
                                C[k][2] = B[n][2]
                                n=n+1; k=k+1;
                        }
                }
        }
        C[0][2] = k-1
}
```

A

4	4	6
1	2	10
1	3	4
1	4	2
3	3	3
4	1	4
4	2	2

B

4	4	6
1	3	8
2	4	3
3	2	2
3	3	9
4	1	2
4	2	7

Traverse the matrix A and B from A[1][1] and B[1][1]
A[1][1] = (1,2,10) and B[1][1] = (1,3,8). As (1,2) is lower entry add the entry to resultant matrix C

A

4	4	6
1	2	10
1	3	4
1	4	2
3	3	3
4	1	4
4	2	2

+

B

4	4	6
1	3	8
2	4	3
3	2	2
3	3	9
4	1	2
4	2	7

=

C

4	4	
1	2	10

Move to the next entry of A matrix A[2][1] and B is still in B[1][1]
A[2][1] = (1,3,4) and B[1][1] = (1,3,8). As both are same entry add the value of two entries 4 + 8 =12 and add to the C matrix

A

4	4	6
1	2	10
1	3	4
1	4	2
3	3	3
4	1	4
4	2	2

+

B

4	4	6
1	3	8
2	4	3
3	2	2
3	3	9
4	1	2
4	2	7

=

C

4	4	
1	2	10
1	3	12

Move both the pointer of A matrix to A[3][1] and B matrix to B[2][1]
A[3][1] = (1,4,2) and B[2][1] = (2,4,3). As (1,4) is the lower entry add the entry to the resultant matrix C

A

4	4	6
1	2	10
1	3	4
1	4	2
3	3	3
4	1	4
4	2	2

+

B

4	4	6
1	3	8
2	4	3
3	2	2
3	3	9
4	1	2
4	2	7

=

C

4	4	
1	2	10
1	3	12
1	4	2

Move to the next entry of A matrix A[4][1] and B is still in B[2][1]
A[4][1] = (3,3,3) and B[2][1] = (2,4,3). As (2,4) is the lower entry add the entry to the resultant matrix C

A

4	4	6
1	2	10
1	3	4
1	4	2
3	3	3
4	1	4
4	2	2

+

B

4	4	6
1	3	8
2	4	3
3	2	2
3	3	9
4	1	2
4	2	7

=

C

4	4	
1	2	10
1	3	12
1	4	2
2	4	3

Move to the next entry of B to B[3][1] and A is still on A[4][1]
A[4][1] = (3,3,3) and B[3][1] = (3,2,2). As (3,2) is the lower entry add the entry to the resultant matrix C

A

4	4	6
1	2	10
1	3	4
1	4	2
3	3	3
4	1	4
4	2	2

+

B

4	4	6
1	3	8
2	4	3
3	2	2
3	3	9
4	1	2
4	2	7

=

C

4	4	
1	2	10
1	3	12
1	4	2
2	4	3
3	2	2

Move to the next entry of B to B[4][1] and A is still on A[4][1]
A[4][1] = (3,3,3) and B[4][1] = (3,3,3). As both are same entry add the value of two entries 3 + 9 =12 and add to the C matrix

A

4	4	6
1	2	10
1	3	4
1	4	2
3	3	3
4	1	4
4	2	2

+

B

4	4	6
1	3	8
2	4	3
3	2	2
3	3	9
4	1	2
4	2	7

=

C

4	4	
1	2	10
1	3	12
1	4	2
2	4	3
3	2	2
3	3	12

Move both the pointer of A matrix to A[5][1] and B matrix to B[5][1]
A[5][1] = (4,1,2) and B[5][1] = (4,1,2). As both are same entry add the value of two entries 4 + 2 =6 and add to the C matrix

A

4	4	6
1	2	10
1	3	4
1	4	2
3	3	3
4	1	4
4	2	2

+

B

4	4	6
1	3	8
2	4	3
3	2	2
3	3	9
4	1	2
4	2	7

=

C

4	4	
1	2	10
1	3	12
1	4	2
2	4	3
3	2	2
3	3	12
4	1	6

Move both the pointer of A matrix to A[6][1] and B matrix to B[6][1]
A[6][1] = (4,2,2) and B[6][1] = (4,2,7). As both are same entry add the value of two entries 2 + 7 =9 and add to the C matrix

A

4	4	6
1	2	10
1	3	4
1	4	2
3	3	3
4	1	4
4	2	2

+

B

4	4	6
1	3	8
2	4	3
3	2	2
3	3	9
4	1	2
4	2	7

=

C

4	4	
1	2	10
1	3	12
1	4	2
2	4	3
3	2	2
3	3	12
4	1	6
4	2	9

A and B matrix reached its end i.e.) no more entries to traverse.
Count the total entries in resultant matrix C and add the value to C[0][2]

A

4	4	6
1	2	10
1	3	4
1	4	2
3	3	3
4	1	4
4	2	2

+

B

4	4	6
1	3	8
2	4	3
3	2	2
3	3	9
4	1	2
4	2	7

=

C

4	4	8
1	2	10
1	3	12
1	4	2
2	4	3
3	2	2
3	3	12
4	1	6
4	2	9

Multiplication

To make our comparisons easier and keep the sorted order, we first calculate the transpose of the 2nd matrix before multiplying the matrices. In order to obtain the resultant matrix, the length of matrices must be traversed, and the relevant multiplied values must be added.

Algorithm:

Step 1: To multiply A and B, generate transpose for second matrix i.e) B to B′

Step 2: Traverse both the matrix A and B′ row-wise

Step 3: Compare 1st column of each row A to the same as B′

Say A[x][1] matches B′[y][1]. Then Store (x:y:A[x][2] * B′[y][2] in resultant matrix C

Step 4: If multiple rows exists in the resultant matrix with values in the 0th and 1st column Add the 2nd column corresponding to the entries and store them as single row in the matrix

Step 5: Sort the resultant matrix

Example:

A

4	4	6
1	2	10
1	3	4
1	4	2
3	3	3
4	1	4
4	2	2

B

4	4	6
1	3	8
2	4	3
3	2	2
3	3	9
4	1	2
4	2	7

Transpose the B matrix

A

4	4	6
1	2	10
1	3	4
1	4	2
3	3	3
4	1	4
4	2	2

B

4	4	6
1	4	2
2	3	2
2	4	7
3	1	8
3	3	9
4	2	3

Traverse through the matrix A and B′ from A[1][1] and check along B′
A[1][1] = (1,2,10) and check the column value of A i.e.) 2 to be the column in B′. The entry (4,2,3) is found at the last entry and hence the row entry and column entry is 1 and 4 with the value as 10 * 3 = 30. The result is stored in C matrix

A

4	4	6
1	2	10
1	3	4
1	4	2
3	3	3
4	1	4
4	2	2

+

B’

4	4	6
1	4	2
2	3	2
2	4	7
3	1	8
3	3	9
4	2	3

=

C

4	4	
1	4	30

Traverse the matrix entry in A to A[2][1] and check along B′. A[2][1] = (1,3,4) and hence check the column value 3 in B′. The entry (2,3,2) is found in B′. The value 4 * 2 = 8 is stored at (1,2,8)

A

4	4	6
1	2	10
1	3	4
1	4	2
3	3	3
4	1	4
4	2	2

+

B’

4	4	6
1	4	2
2	3	2
2	4	7
3	1	8
3	3	9
4	2	3

=

C

4	4	
1	4	30
1	2	8

Further it is checked to find any column has value 3. (3, 3, 9) is found in B′. The value 4 * 9 = 36 is stored as (1, 3, 36) to the resultant matrix.

A

4	4	6
1	2	10
1	3	4
1	4	2
3	3	3
4	1	4
4	2	2

+

B’

4	4	6
1	4	2
2	3	2
2	4	7
3	1	8
3	3	9
4	2	3

=

C

4	4	
1	4	30
1	2	8
1	3	36

Traverse the matrix entry in A to A[3][1] and check along B′. A[3][1] = (1,4,2) and hence check the column value 4 in B′. The entry (1,4,2) is found in B′. The value 2 * 2 = 4 is stored at (1,1,4)

A

4	4	6
1	2	10
1	3	4
1	4	2
3	3	3
4	1	4
4	2	2

+

B’

4	4	6
1	4	2
2	3	2
2	4	7
3	1	8
3	3	9
4	2	3

=

C

4	4	
1	4	30
1	2	8
1	3	36
1	1	4

Further it is checked to find any column has value 4. (2, 4,7) is found in B′. The value 2 * 7 = 14 is stored as (1, 2, 14) to the resultant matrix.

A

4	4	6
1	2	10
1	3	4
1	4	2
3	3	3
4	1	4
4	2	2

+

B’

4	4	6
1	4	2
2	3	2
2	4	7
3	1	8
3	3	9
4	2	3

=

C

4	4	
1	4	30
1	2	8
1	3	36
1	1	4
1	2	14

Traverse the matrix entry in A to A[4][1] and check along B′. A[3][1] = (3,3,3) and hence check the column value 3 in B′. The entry (2,3,2) is found in B′. The value 3 * 2 = 6 is stored at (3,2,6)

Further it is checked to find any column has value 3. (3, 3, 9) is found in B′. The value 3 * 9 = 27 is stored as (3, 3, 27) to the resultant matrix.

Traverse the matrix entry in A to A[5][1] and check along B′. A[5][1] = (4,1,4) and hence check the column value 1 in B′. The entry (3, 1, 8) is found in B′. The value 4 * 8 = 32 is stored at (4, 3, 32)

Traverse the matrix entry in A to A[6][1] and check along B′. A[6][1] = (4,2,2) and hence check the column value 2 in B′. The entry (4, 2, 3) is found in B′. The value 2 * 3 = 6 is stored at (4, 4, 6)

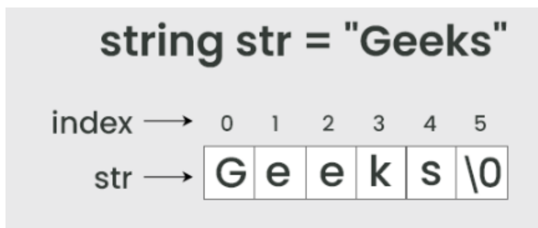
Check if there are any repeated entries. If so, add. The entry (1, 2, 8) and (1, 2, 14) is found and hence added as (1, 2, 22).

Sort the entries in resultant matrix

Count the total entries and store in C[0][2] as 8

STRING AS ABSTRACT DATA TYPE

In data structures, a string is a sequence of characters used to represent text. Strings are commonly used for storing and manipulating textual data in computer programs. They can be manipulated using various operations like concatenation, substring extraction, and comparison.



String is considered a data type in general and is typically represented as arrays of bytes (or words) that store a sequence of characters. String is defined as an array of characters. The difference between a character array and a string is the string is terminated with a special character ‘\0’. Some examples of strings are: “geeks” , “for”, “geeks”, “GeeksforGeeks”, “Geeks for Geeks”, “123Geeks”, “@123 Geeks”.

String Data Type:

In most programming languages, strings are treated as a distinct data type. This means that strings have their own set of operations and properties. They can be declared and manipulated using specific string-related functions and methods.

String Operations:

Strings support a wide range of operations, including concatenation, substring extraction, length calculation, and more. These operations allow developers to manipulate and process string data efficiently.

Below are fundamental operations commonly performed on strings in programming.

Concatenation: Combining two strings to create a new string. String concatenation is the process of joining two strings end-to-end to form a single string.

Examples
Input: str1 = “Hello”, str2 = “World”
Output: “HelloWorld”
Explanation: Joining “Hello” and “World” results in “HelloWorld”.
Input: str1 = “Good”, str2 = “Morning”
Output: “GoodMorning”
Explanation: Joining “Good” and “Morning” results in “GoodMorning”
The strcat() function in C is used for string concatenation. It will append a copy of the source string to the end of the destination string.
Syntax

char* strcat(char* dest, const char* src);

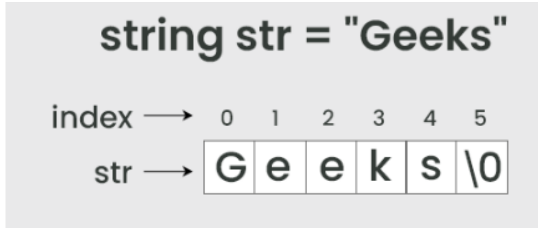
The terminating character at the end of dest is replaced by the first character of src.
Parameters
dest: Destination string
src: Source string
Return value
The strcat() function returns a pointer to the dest string.

Length: Determining the number of characters in a string. The length of a string is the number of characters in it excluding the null character (‘\0’).
Examples:
Input: str[] = “Hello”
Output: 5
Explanation: The string “Hello” has 5 characters.
Input: str[] = “GeeksforGeeks”
Output: 13
Explanation: The string “GeeksforGeeks” has 13 characters.
The **strlen()** function calculates the length of a given string. It doesn’t count the null character ‘\0’.
Syntax

int strlen(const char *str);

Parameters
str: It represents the string variable whose length we have to find.
Return Value
strlen() function in C returns the length of the string.

Access: Accessing individual characters in a string by index.



Substring: Extracting a portion of a string. A substring is a contiguous sequence of characters within a String.
Input: Str =”the”, pos=1, len=2
Output: “th”
Explanation: substrings will be: “”, “t”, “h”, “e”, “th”, “he”, “the”.
Input: Str =”geeks”, pos=3, length=3
Output: “eks”
Explanation: substrings are: “”, ” g”, “e”, “e”, “k”, “s”, “ge”, “ee”, “ek”, “ks”, “gee”, “eek”, “eks”, “geek”, “eeks”, “geeks”.
Steps:

- Create a character array to store the substring.
- Iterate from the given position for the given length to generate the substring required.
- Then store each character in the character array and print the substring.

Comparison: Comparing two strings to check for equality or order. C strcmp() is a built-in library function that is used for string comparison. This function takes two strings (array of characters) as arguments, compares these two strings lexicographically, and then returns 0,1, or -1 as the result. It is defined inside <string.h> header file with its prototype as follows:
Syntax of strcmp() in C

strcmp(first_str, second_str);

Parameters of strcmp() in C
This function takes two strings (array of characters) as parameters:
first_str: First string is taken as a pointer to the constant character (i.e. immutable string).
second_str: Second string is taken as a pointer to a constant character.
Return Value

- If str1 is less than str2, the return value is less than 0.
- If str1 is greater than str2, the return value is greater than 0.
- If str1 is equal to str2, the return value is 0.

Search: Finding the position of a specific substring within a string. The strchr() function in C is a predefined function used for string handling. This function is used to find the first occurrence of a character in a string.

Syntax

char *strchr(const char *str, int c);

Parameters

str: specifies the pointer to the null-terminated string to be searched in.

ch: specifies the character to be searched for.

Here, str is the string and ch is the character to be located. It is passed as its int promotion, but it is internally converted back to char.

Return Value

It returns a pointer to the first occurrence of the character in the string.
