

|  |
| --- |
| [Type the document subtitle] | Student |

|  |  |
| --- | --- |
| CIC-255 | DATA STRUCTURES LAB |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **EXP. NO.** | **EXPERIMENT NAME** | **PAGE NO.** | **DATE** | **SIGN/**  **REMARKS** |
|  |  |  |  |  |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **EXP. NO.** | **EXPERIMENT NAME** | **PAGE NO.** | **DATE** | **SIGN/**  **REMARKS** |
|  |  |  |  |  |

**EXPERIMENT 1**

**AIM:** Implement sparse matrix using array. Description of program:

1. Read a 2D array from the user.
2. Store it in the sparse matrix form, use array of structures.
3. Print the final array.

**ALGORITHM:**

1. Get a sparse matrix inputted from a user.
2. Loop through the sparse matrix for i,j index
3. Set element = sparse matrix[i][j]
4. If element != 0, store the value of i, j and element in an array of structure

[End of Step 2 loop]

1. Print the array of structure using a loop
2. Exit

A sparse matrix is a matrix which has a high number of 0 elements. This causes a lot of space to be wasted by storing the same value.

So for storing a sparse matrix we use this algorithm where, we traverse through the sparse matrix and find out the nonzero elements. We then input the non zero elements with their row and columns in new array.This method is called 3 tuple notation.

Since we are storing 3x data for non zero elements, it is preferred to convert sparse matrix to 3 tuple notation when the non zero to zero ratio is less than 1/3.

**CODE:**

// Header files

#include <stdio.h>

#include <stdlib.h>

// Struct that will store our non zero data

struct DataElement

{

int value;

int row;

int column;

};

// Prototyping the functions

int \*input2D\_matrix1D(int \*, int \*);

struct DataElement \*shredder(int[], int, int, int \*);

void printCompactMatrix(struct DataElement \*, int);

int main()

{

// Getting a user inputted 2D sparse array that is passed as 1D through the function

int rows, columns;

int \*sparse\_matrix1D = input2D\_matrix1D(&rows, &columns);

// Converting the sparse array into a compact array (array of structure)

int size;

struct DataElement \*compact\_matrix = shredder(sparse\_matrix1D, rows, columns, &size);

// Printing the compact array

printCompactMatrix(compact\_matrix, size);

return 0;

}

struct DataElement \*shredder(int sparse\_matrix1D[], int rows, int columns, int \*compact\_size)

{

// This function converts sparse matrix to compact matrix. Returns struct array and array size.

int(\*sparse\_matrix)[rows] = (int(\*)[rows])sparse\_matrix1D; // Converts 1D array into 2D array using typecasting

// Calculates the number of non-zero elements in the matrix

int filled\_count = 0;

for (int i = 0; i < rows; i++)

for (int j = 0; j < columns; j++)

if (sparse\_matrix[i][j] != 0)

filled\_count++;

// Creates a struct array with length as of the number of non-zero elements

struct DataElement \*compact\_matrix = (struct DataElement \*)malloc(filled\_count \* sizeof(struct DataElement));

// Add structs to the struct array

filled\_count = 0;

for (int i = 0; i < rows; i++)

for (int j = 0; j < columns; j++)

if (sparse\_matrix[i][j] != 0)

{

compact\_matrix[filled\_count].value = sparse\_matrix[i][j];

compact\_matrix[filled\_count].row = i;

compact\_matrix[filled\_count].column = j;

filled\_count++;

}

// Returns the array and its size

\*compact\_size = filled\_count;

return compact\_matrix;

}

int \*input2D\_matrix1D(int \*rows, int \*columns)

{

// This function takes input from the user and creates an array from that

printf("Enter the number of Rows: ");

scanf("%d", rows); // It also returns the value back because pointer

printf("Enter the number of Columns: ");

scanf("%d", columns);

// Creates an array with length as if the 2D matrix was converted to 1D row wise

// This is done since you cannot return a variable length multidimensional array

int \*matrix = (int \*)malloc((\*rows) \* (\*columns) \* sizeof(int));

for (int i = 0; i < \*rows; i++)

for (int j = 0; j < \*columns; j++)

{

printf("Enter the number at position (%d,%d): ", i, j);

scanf("%d", matrix + i \* (\*columns) + j);

}

return matrix;

}

void printCompactMatrix(struct DataElement \*matrix, int size)

{

// Prints the compact matrix using a for loop

printf("\nCompact Matrix \n");

for (int i = 0; i < size; i++)

printf("Element No. %d, Value: %d Row: %d Column: %d \n", i, matrix[i].value, matrix[i].row, matrix[i].column);

}

**OUTPUT:**

PS C:\Users\admin\Documents\Saksham Gupta\MSIT> ./a.exe

Enter the number of Rows: 3

Enter the number of Columns: 3

Enter the number at position (0,0): 0

Enter the number at position (0,1): 0

Enter the number at position (0,2): 0

Enter the number at position (1,0): 0

Enter the number at position (1,1): 2

Enter the number at position (1,2): 4

Enter the number at position (2,0): 0

Enter the number at position (2,1): 0

Enter the number at position (2,2): 1

Compact Matrix

Element No. 0, Value: 2 Row: 1 Column: 1

Element No. 1, Value: 4 Row: 1 Column: 2

Element No. 2, Value: 1 Row: 2 Column: 2

**EXPERIMENT 2**

**AIM:** Implement linear search and binary search on an array.

**ALGORITHM:**

LINEAR SEARCH:

1. Get the number to be searched inputted from the user and SET it to item.
2. Loop through the array with index i
3. If item == array[i] , element is present in the array,

print the element is present,

exit

[End of Step 2 loop]

1. Else element is not present in the array,

Print the element is not present

Exit

The working behind a linear search is that we traverse through each element in the list and check if that’s the element we needed to search for.

Advantage: Easy to implement

Disadvantage: High computational order

BINARY SEARCH:

1. Get the number to be searched inputted from the user and SET it to item.
2. Sort the array.
3. Set lo = 0, hi = size of array - 1 // first and last element
4. Loop till hi –lo >0 //they are not coinciding or consecutive
5. Set mid =( hi + lo)/2
6. If array[mid] < item, lo = mid+1 //element in second half of list
7. Else hi = mid // element in first half of list

[End of Step 4 loop]

1. if item == array[hi] or array[low], element is present in the array,

Print the element is present

Exit

1. Else element is not present in the array,

Print the element is not present

Exit

The working behind binary search is to have a sorted list and then dividing the list into equal halves and checking in which half the item lies in and continue to divide it further until we are homed on one/two element(s) in the list.

The basic steps to perform Binary Search are:

* Begin with the mid element of the whole array as a search key.
* If the value of the search key is equal to the item then return an index of the search key.
* Or if the value of the search key is less than the item in the middle of the interval, narrow the interval to the lower half.
* Otherwise, narrow it to the upper half.
* Repeatedly check from the second point until the value is found or the interval is empty.

**CODE:**

// Header files

#include <stdio.h>

int list[10] = {1, 4, 5, 10, 30, 15, 2, -1, -15, 20}; // Array which is used for searching

int size = 10; // Array's size

// Prototyping the functions

int\* bubble\_sort(int\*,int);

void linear\_search(int[],int,int);

void binary\_search(int[],int,int);

int main()

{

// Getting the element to be searched from the user

int search\_element;

printf("Input the search element: ");

scanf("%d",&search\_element);

// Doing searches using functions

linear\_search(list,size,search\_element);

binary\_search(list,size,search\_element);

return 0;

}

void linear\_search(int list[], int size, int search\_element)

{

//This function uses linear search to find the element in list

for (int i = 0; i < size; i++) //Traversing through the list

{

if (list[i] == search\_element) //If element found

{

printf("Found the element %d at position %d using linear search.\n", search\_element, i);

return;

}

}

// If foor loop is Completed and element not found

printf("Could not find the element %d in the list using linear search.\n", search\_element);

return;

}

void binary\_search(int list[], int size, int search\_element)

{

// This function searches for an element using binary search

int\* sorted\_list = bubble\_sort(list,size); // binary search uses a sorted list

int lo = 0, hi= size-1; // first and last index

int mid;

// while indexes not coinciding or consecutive

while (hi - lo > 1) {

int mid = (hi + lo) / 2;

if (list[mid] < search\_element) { //element is in the second half of the list

lo = mid + 1;

}

else { //element is in the first half of the list

hi = mid;

}

}

// Loop finished with finding the element

if (list[lo] == search\_element || list[hi] == search\_element)

printf("Found the element %d in the list using binary search.", search\_element);

// Loop finished without finding the element

else

printf("Could not find the element %d in the list using binary search.\n", search\_element);

}

int \*bubble\_sort(int list[], int size)

{

//This function sorts the list using bubble sort

for (int i = 0; i < size - 1; i++)

{

for (int j = 0; j < size - i - 1; j++)

{

if (list[j] > list[j+1])

{

int swap\_var = list[j];

list[j] = list[j + 1];

list[j + 1] = swap\_var;

}

}

}

return list;

}

**OUTPUT:**

* **Test Case 1:** Element in the list

PS C:\Users\admin\Documents\Saksham Gupta\MSIT> ./a.exe

Input the search element: 10

Found the element 10 at position 3 using linear search.

Found the element 10 in the list using binary search.

PS C:\Users\admin\Documents\Saksham Gupta\MSIT> ./a.exe

Input the search element: 30

Found the element 30 at position 4 using linear search.

Found the element 30 in the list using binary search.

* **Test Case 2:** Element not in the list

PS C:\Users\admin\Documents\Saksham Gupta\MSIT> ./a.exe

Input the search element: 50

Could not find the element 50 in the list using linear search.

Could not find the element 50 in the list using binary search.