|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **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.

Example:

1. Take a sparse matrix A =
2. Take a struct(row, column, value) array B
3. Loop through the matrix taking i, j index
4. i = 0, j = 0 , element = A[0][0] = 2
5. element != 0, store struct(0, 0, 2) in B
6. I = 0, j = 1, element = A[0][1] = 0
7. Element == 0, skip
8. I = 1, j = 0, element = A[1][0] = 0
9. Element == 0, skip
10. I = 1, j = 1, element = A[1][1] = 1
11. Element != 0, store struct(1, 1, 1) in B
12. Print B
13. Exit

**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 >1 //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.

EXAMPLE:

A = [1,2,3,4,5,6,7]

To find 3 in list A

Linear Search:

1. Item = 3
2. Loop through A with index i
3. I = 0, a[0] = 1, 1!=item
4. I = 1, a[1] = 2, 2!=item
5. I = 2, a[2] = 3, 3==item, print item present and exit

Binary search

1. Item = 3
2. Sort array A, A is already sorted
3. Lo = 0, hi = size of A -1 = 7-1 = 6
4. Loop till hi - lo > 1
5. Mid = (hi – lo)//2 = 3
6. A[mid] = A[3] = 4, 4>item, therefore hi =mid =3
7. Hi – lo = 3 > 1, loop continues
8. Mid = (hi-lo)//2 = 1
9. A[mid] = A[1] = 2, 2<item, therefore lo =mid+1 =2
10. Hi – lo = 1 not > 1, loop ends
11. (A[2] == 3) or (A[3] ==3} is true
12. Print element is present in the 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.

**EXPERIMENT 3**

**AIM:** Create a linked list with nodes having information about a student and perform:

1. Insert a new node at a specified position.
2. Delete a node with the roll number of a student specified.
3. Reversal of that linked list.

**ALGORITHM:**

**INSERTING NEW NODE AT SPECIFIC POSITION**

1. [check for overflow]

Ptr = (struct node \*) malloc (sizeof(struct node));

If ptr = null

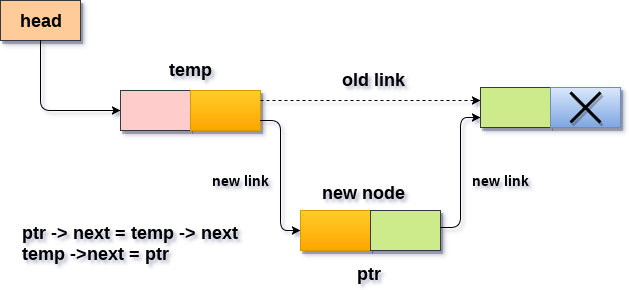
Print overflow   
exit

1. Ptr->data = item;
2. Set temp = head
3. Set i = 0
4. Repeat until i = desired node index
5. Temp = temp → next
6. If temp = null

Print "desired node not present"  
 Exit

[ end of loop]

1. Ptr → next = temp → next
2. Temp → next = ptr
3. Exit



**DELETING A SPECIFIC NODE**

1. [check for underflow]

If head = null

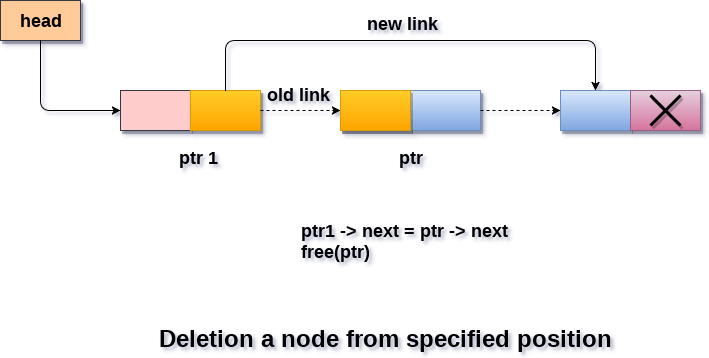
Print underflow   
Exit

1. Set ptr = head
2. Set i = 0
3. Repeat until i = desired node index
4. Set ptr1= ptr
5. ptr = ptr → next
6. If ptr = null

Print "desired node not present"  
 Exit

[ end of loop]

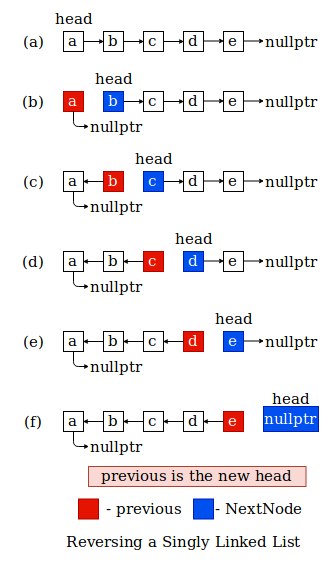
1. Ptr1 → next =ptr → next
2. Free(ptr)
3. Exit



**REVERSAL OF LINKED LIST**

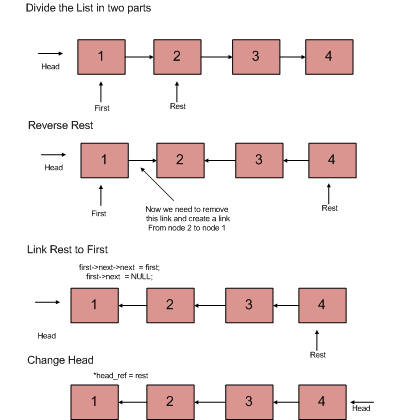
Iterative method

1. Initialize three pointers prev as NULL, curr as head and next as NULL.
2. Iterate through the linked list.   
   // Before changing next of current,   
   // store next node   
   next = curr->next  
   // Now change next of current   
   // This is where actual reversing happens   
   curr->next = prev   
   // Move prev and curr one step forward   
   prev = curr   
   curr = next



Recursive Method

1. Divide the list in two parts - first node and rest of the linked list.
2. Call reverse for the rest of the linked list.
3. Link rest to first.
4. Fix head pointer



**OUTPUT:**

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

Created Linked list:

Roll No: 14460 Name: Sunil Shetty Class: 6-B

Roll No: 14461 Name: Rajiv Gandhi Class: 7-F

Roll No: 14462 Name: Ajay Devgan Class: 12-D

Linked list after insterting a student data at index 2:

Roll No: 14460 Name: Sunil Shetty Class: 6-B

Roll No: 14461 Name: Rajiv Gandhi Class: 7-F

Roll No: 14463 Name: Ritesh Singh Class: 11-B

Roll No: 14462 Name: Ajay Devgan Class: 12-D

Linked list after deleting a student with roll number 14461:

Roll No: 14460 Name: Sunil Shetty Class: 6-B

Roll No: 14463 Name: Ritesh Singh Class: 11-B

Roll No: 14462 Name: Ajay Devgan Class: 12-D

Reversed Linked list through loop:

Roll No: 14462 Name: Ajay Devgan Class: 12-D

Roll No: 14463 Name: Ritesh Singh Class: 11-B

Roll No: 14460 Name: Sunil Shetty Class: 6-B

Reversed Linked list through recursion:

Roll No: 14460 Name: Sunil Shetty Class: 6-B

Roll No: 14463 Name: Ritesh Singh Class: 11-B

Roll No: 14462 Name: Ajay Devgan Class: 12-D