**PSG COLLEGE OF TECHNOLOGY, COIMBATORE-04**

**DEPARTMENT OF APPLIED MATHEMATICS AND COMPUTATIONAL SCIENCES**

**II Semester MSc Software Systems**

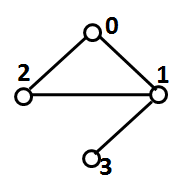
**18XW26 Data Structures Lab**

**Data Structures Lab –Linked List**

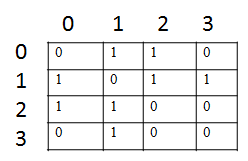
**Problem sheet - 9**

1. Given a list, split it into two sublists — one for the front half, and one for the back half. If the number of elements is odd, the extra element should go in the front list. So FrontBackSplit() on the list {2, 3, 5, 7, 11} should yield the two lists {2, 3, 5} and {7, 11}. Getting this right for all the cases is harder than it looks. You should check your solution against a few cases (length = 2, length = 3, length=4) to make sure that the list gets split correctly near the short-list boundary conditions. If it works right for length=4, it probably works right for length=1000. You will probably need special case code to deal with the (length <2) cases.
2. Write a RemoveDuplicates() function which takes a list sorted in increasing order and deletes any duplicate nodes from the list. Ideally, the list should only be traversed once.
3. Write a function AlternatingSplit() that takes one list and divides up its nodes to make two smaller lists. The sublists should be made from alternating elements in the original list. So if the original list is {a, b, a, b, a}, then one sublist should be {a, a, a} and the other should be {b, b}. You may want to use MoveNode() as a helper. The elements in the new lists may be in any order (for some implementations, it turns out to be convenient if they are in the reverse order from the original list.)
4. Given two lists, merge their nodes together to make one list, taking nodes alternately between the two lists. So ShuffleMerge() with {1, 2, 3} and {7, 13, 1} should yield {1, 7, 2, 13, 3, 1}. If either list runs out, all the nodes should be taken from the other list.
5. Write a SortedMerge() function that takes two lists, each of which is sorted in increasing order, and merges the two together into one list which is in increasing order. SortedMerge() should return the new list.
6. It is possible to use more than one data structures together to give efficient solution for many real time applications. The following is the one of the situations where array is used with linked list to form a well organized data representation.

Graphs are usually represented as adjacency matrices. Consider the following graph.



The adjacency matrix of the above graph is,

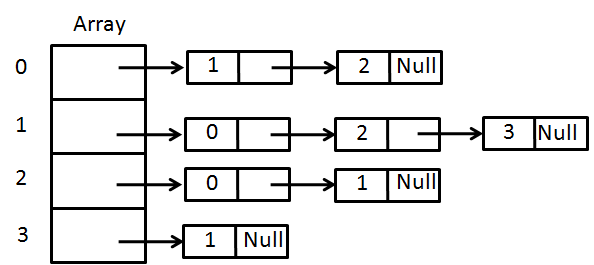


To store adjacency matrix for a graph with n nodes required n X n elements to represent their link information and it is worst representation for sparse graphs, since it requires more memory to store 0 (no link between corresponding nodes). For such situations, linked list is the best idea to represent the graph.

**Adjacency List:**

An array of linked lists is used. Size of the array is equal to number of vertices. Let the array be Array[]. An entry Array[i] represents the linked list of vertices adjacent to the **i**th vertex.

The adjacency linked list representation of the above graph is as follows,



Write a C/C++ program to read an adjacency matrix of a graph, and create an adjacency list for that graph. For this, you have to create an array of singly linked lists, where each list represents the neighbors of a particular node. Display the contents of the linked list by starting the execution from 0th index of the array.

For example, for the above graph, the output is as follows.

0 - 1 2

1 - 0 2 3

2 - 0 1

3 - 1