Date : / /
Page No. 1

MCSL - 209

Q-1. Write an algorithm and program in 'C' language to merge two sorted linked lists. The resultant linked list should be sorted.

Ans-1. Algorithm:

- If head1 is null, return head2.
- If head2 is null, return head1.
- Compare head1.data and head2.data.
- If head1.data <= head2.data.

=> set head = head1

=> set head.next = merge(head1.next, head2)

Else:

=> set head = head2

=> set head.next = merge(head1, head2.next)

- Return head.

Program to merge two sorted linked list:

Input:

Head1 = 5 => 10 => 15 => 40

Head2 = 2 => 3 => 20

```
#include <stdio.h>
#include <stdlib.h>
struct node
{
int data;
struct node *next;

};
struct node *sortedmerge (struct node *head1, struct node *
```

```
head2)
{
//base cases
if(head1 == null)
  return head2;
if(head2 == null)
  return head1;

//recursive merging based on smaller value
if(head1->data <= head2->data)
{
head1->next = sorted merge(head1->next, head2);
  return head1;
}
else
{
head2->next = sorted merge(head1, head2->next);
  return head2;
  }
}
void print list(struct node * curr)
{
while(curr != null)
{
printf("%d", curr->data);
```

```
if (curr->next!=null)
{
printf("->");
}
curr=curr->next;
}
printf("\n");
}
struct node *newnode=(struct node*)malloc(sizeof(struct node));
newnode->data=data;
newnode->next=null;
return newnode;
}
int main()
{
struct node *head1=create node(5);
head1->next=create node(10);
head1->next->next=create node(15);
head1->next->next->next=create node(40);
```

```
Struct node *head2 = Create node (2);
head2 -> next = Create node (3);
head2 -> next -> next = Create node (20);
Struct node *res = sorted merge(head1, head2);
Print list(res);
return 0;

}
```

Output

2 -> 3 -> 5 -> 10 -> 15 -> 20 -> 40

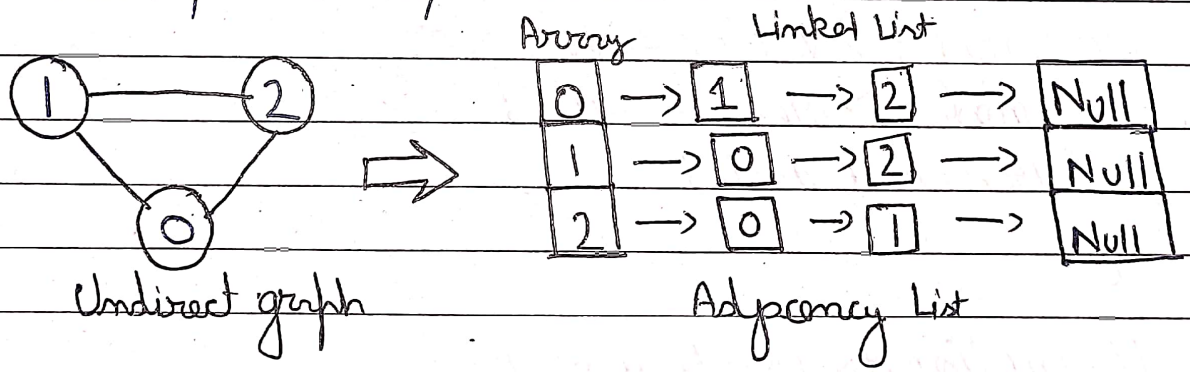
Q-2 Write an algorithm and a program in 'C' language to insert and delete edges in an adjacency list representation of an undirected graph. Make assumptions, if necessary.

Ans-2 Create a struct graph that will have the following data members:

- num vertices: represent the total number of vertices in a graph.
- adjlist: a double pointer to represent an array of lists.
- is directed: a flag variable to denote whether the graph is directed or not.
- Create an array of linked lists adjlist of size equal to the number of vertices in the graph.
- Initially point each array element to null until the graph is intialized.
- Implement a function add edge (src, dest) to add an edge for the graph.
- Create a new node with data equal to dest and add it in the

adjlist [src].

- If the graph is undirected then create a new node with data to src and add it in the adjlist [dest].
- Implement a function printgraph () to print the adjacency list for the adj graph.
- Iterate through each vertex of the graph present in the adjacency list.
- Print the vertex.
- Initialise a temporary pointer tead of the list present at adjlist [vertex].
- While teme is not equal to null:
- Print temp -> data.
- Move temp to temp -> next.



Graph Representation of Undirect graph to Adjacency List

Program to insert/delete edges in an adjacency list representation of an undirected graph

```
// C program to implement adjacency list in C++
#include <stdio.h>
#include <stdlib.h>
// Structure to represent a node in the adjacency list
```

```
struct Node {
  int Vertex;
Struct Node* next;
};
// Structure to represent the graph
Struct graph {
int num Vertices;
Struct node ** adjlists;
int is directed;

};
//Function to create a new node
struct node* new node = malloc (size of (struct node));
new node -> Vertex = V;
new node -> next = null;
return new node;

}
//function to create a graph
struct graph * create graph (int vertices, int is directed)

{
struct graph * ~~creat~~ graph (~~int~~ = malloc (size of (struct graph)),
graph -> num vertices = vertices;
graph -> is directed = is directed;

//create an array of adjacency lists
```

```
graph->adjlists = malloc(vertices * sizeof(struct node*));

//initialize each adjacency list as empty
for(int i=0; i<vertices; i++)
{
graph->adjlist[i] = NULL;
}
return graph;
}
//Function to add an edge to the graph
void addedge(struct graph* graph, int src, int dest)
{
//Add edge from src to dest
Struct node* newnode = createnode(dest);
newnode->next = graph->adjlist[src];
graph->adjlists[src] = newnode;

//if the graph is undirected, add an edge from dest to src as
well
if(!graph->isdirected)
{
newnode = createnode(src);
```

Date : / /
Page No. 8

```
newNode->next = graph->adjLists[dest];
graph->adjLists[dest] = newNode;

    }
}
//Function to print the adjacency list representation of the
graph void printGraph(struct graph *graph)
{
printf("Vertex: Adjacency list\n");
for(int v=0; v< graph->numVertices; v++)
{
struct node* temp = graph->adjLists[v];
Printf("%d --->", v);
while(temp)
{
printf("%d->", temp->vertex);
temp = temp->next;
}
printf("null \n");
}
}
int main()
```

```
{
// Create an undirected graph with 3 vertices
struct graph *undirectedgraph = Create graph (3, 0);

// Add edges to the undirected graph

add edge(undirected graph, 0, 1);
add edge (undirected graph, 0, 2);
add edge (undirected graph, 1, 2);

Print ("Adjacency list for undirected graph: \n");
Print graph (Undirected graph);

// Create a directed graph with 3 vertices
struct graph *directed graph = Create graph (3, 1);


// Add edges to the directed graph
add edge (directed grap, 1, 0);
add edge (directed graph, 1, 2);
add edge (directed graph, 2, 0);


Print f ("\n Adjacency list for directed graph: \n");

Print graph (directed graph);
```

```
return 0;
}
```

Out put

Adjacency list for undirected graph:

Vertex: Adjacency list

0 ---> 2 -> 1 -> NULL

1 ---> 2 -> 0 -> NULL

2 ---> 1 -> 0 -> NULL