Comp Practical

1.Adjacency matrix

```
#include <stdio.h>
#define MAX_VERTICES 100
int main()
{
   int vertices, edges;
   int adjacencyMatrix[MAX_VERTICES][MAX_VERTICES] = {0};
   printf("Enter the number of vertices: ");
   scanf("%d", &vertices);
   printf("Enter the number of edges: ");
   scanf("%d", &edges);
   printf("Enter the edges (u v) where u and v are vertex indices (0 to %d): \n", vertices - 1);
   for (int i = 0; i < edges; i++)
   {
      int u,v;
      scanf("%d %d",&u, &v);
      adjacencyMatrix[u][v] = 1;
      adjacencyMatrix[v][u] = 1;
   }
   printf("Adjancy Matrix: \n");
   for(int i=0 ; i<vertices ; i++)
   {
      for(int j=0; j<vertices; j++)
      {
         printf("%d", adjacencyMatrix[i][j]);
      }
      printf("\n");
   }
```

```
    return 0;
}
```

2. **Adjacency List**

```
#include <stdio.h>
#include <stdlib.h>
struct Node
{
    int v;
    struct Node* next;
};
struct Graph
{
    int V;
    struct Node** adj;
};
struct Node* createNode(int v)
{
    struct Node* newNode=(struct Node*)malloc(sizeof(struct Node));
    newNode->v = v;
    newNode->next = NULL;
    return newNode;
}
struct Graph* createGraph(int V)
{
    struct Graph* graph = (struct Graph*)malloc(sizeof(struct Node*));
    graph->V = V;
    graph->adj = (struct Node*)malloc(V* sizeof(struct Node*));
    for(int i = 0; i < V; i++)
    {
    graph->adj[i] = NULL;
```

```
    }
    return graph;
}
void addEdge(struct Graph* graph,int src, int dest)
{
    struct Node* newNode = createNode(dest);
    newNode->next = graph->adj[src];
    graph->adj[src] = newNode;
    newNode = createNode(src);
    newNode->next = graph->adj[dest];
    graph->adj[dest] = newNode;
}
void displayGraph(struct Graph*graph)
{
    for(int i = 0; i < graph->V; i++)
    {
        struct Node* temp = graph->adj[i];
        printf("Vertex %d: ", i);
        while (temp)
        {
            printf("%d -> ", temp->v);
            temp = temp->next;
        }
        printf("NULL\n");
    }
}
int main()
{
    int V, E, src, dest;
    printf("Enter the number of vertices: ");
    scanf("%d",&V);
```

```
    printf("Enter the number of edges: ");
    scanf("%d", &E);
    struct Graph* graph = createGraph(V);
    printf("Enter the edges(src dest):\n");
    for (int i = 0; i < E; i++)
    {
        scanf("%d %d",&src, &dest);
        addEdge(graph, src, dest);
    }
    printf("\n Adjacency List: \n");
    displayGraph(graph);
    return 0;
}
```

3.Adjacency Matrix Indegree Outdegree Total degree

```
#include <stdio.h>
#define MAX 10
int adj[MAX][MAX];
int v, e;
void init()
{
    for(int i = 0; i < v; i++)
    {
        for(int j = 0; j < v; j++)
        {
            adj[i][j] = 0;
        }
    }
}
void inputEdges()
{
```

```
    int src, dest;
    for(int i = 0; i < e; i++)
    {
        printf("Enter edges(src dest): ");
        scanf("%d %d", &src, &dest);
        adj[src][dest] = 1;
    }
}
void printInDegree()
{
     printf("In-Degree:\n");
     for(int i = 0; i < v; i++)
     {
         int in_deg = 0;
         for(int j = 0; j < v; j++)
         {
             in_deg += adj[j][i];
         }
         printf("Vertex %d: %d\n", i, in_deg);
     }
}
void printOutDegree()
{
    printf("Out-degrees:\n");
    for(int i = 0; i < v; i++)
    {
        int out_deg = 0;
        for(int j = 0; j < v; j++)
        {
            out_deg += adj[i][j];
        }
```

```
        printf("Vertex %d: %d\n", i, out_deg);
    }
}
void printTotalDegree()
{
    printf("Total-degrees:\n");
    for(int i = 0; i < v; i++)
    {
        int in_deg = 0, out_deg = 0;
        for(int j = 0; j < v; j++)
        {
            in_deg += adj[j][i];
            out_deg += adj[i][j];
        }
        printf("Vertex %d: %d\n", i, in_deg + out_deg);
    }
}
int main()
{
    printf("Enter the number of vertices: ");
    scanf("%d",&v);
    printf("Enter the number of edges: ");
    scanf("%d", &e);

    init();
    inputEdges();

    printInDegree();
    printOutDegree();
    printTotalDegree();
```

```
   return 0;
}
```

4.Prim's Algorithm

```
#include<stdio.h>
#include<limits.h>

#define V 5

int minkey(int key[], int mstSet[])
    {
      int min = INT_MAX , min_index;

      for(int v=0; v < V; v++)
        {
            if(mstSet[v] == 0 && key[v] < min)
                {
                    min = key[v];
                    min_index = v;
                }
        }
         return min_index;
    }

void primMST(int graph[V][V])
    {
        int parent[V];
        int key[V];
        int mstSet[V];

        for(int i = 0; i < V; i++)
```

```
        {
            key[i] = INT_MAX;
            mstSet[i] = 0 ;
        }

    key[0] = 0;
    parent[0] = -1;

    for(int count = 0; count < V-1; count++)
        {
            int u = minkey(key,mstSet);
            mstSet[u] = 1;
            for(int v = 0; v < V; v++)
                {
                    if(graph[u][v] && mstSet[v] == 0 && graph[u][v] < key[v])
                        {
                            parent[v] = u;
                            key[v] = graph[u][v];
                        }
                }
        }

        printf("edges \t weight \n");
        for(int i =1 ; i < V ; i++)
            {
                printf("%d - %d \t %d \n",parent[i], i , graph[i][parent[i]]);
            }
    }

int main()
    {
```

```
        int graph[V][V]= {
                        {0,2,0,6,0},
                        {2,0,3,8,5},
                        {0,3,0,0,7},
                        {6,8,0,0,9},
                        {0,5,7,9,0}
                       };
        primMST(graph);

        return 0;
    }
```

5.BST Inorder Preorder Postorder

```
#include <stdio.h>
#include <stdlib.h>

struct Node
{
    int data;
    struct Node* left;
    struct Node* right;
};
struct Node* createNode(int data)
{
    struct Node* NewNode=(struct Node*)malloc(sizeof(struct Node));
    NewNode -> data= data;
    NewNode -> left= NULL;
    NewNode -> right=NULL;
    return NewNode;
}
struct Node* insert(struct Node* root, int data)
```

```
{
    if(root==NULL)
    {
    return createNode(data);
    }
    if (data < root -> data)
    {
    root->left = insert(root-> left,data);
    }
    else
    {
    root->right = insert(root -> right,data);
    }
    return root;
}
void inorder(struct Node* root)
{
   if (root!=NULL)
   {
      inorder(root->left);
      printf("%d",root->data);
      inorder(root->right);
    }
}
void preorder(struct Node * root)
{
   if (root!=NULL)
   {
      printf("%d",root->data);
      preorder(root->left);
      preorder(root->right);
```

```
    }
}
void postorder(struct Node * root)
{
   if (root!=NULL)
   {
      postorder(root->left);
      postorder(root->right);
      printf("%d",root->data);
    }
}
void freeTree(struct Node* root)
{
   if (root != NULL)
   {
      freeTree(root->left);
      freeTree(root->right);
      free(root);
   }
}
int main()
{
   struct Node* root = NULL;
   int choice,data;

   do
   {
      printf("\nMenu:\n");
      printf("1. Insert\n");
      printf("2. Inorder Traversal\n");
      printf("3. Preorder Traversal\n");
```

```
printf("4. Postorder Traversal\n");
printf("5. Exit\n");
printf("Enter your choice: ");
scanf("%d", &choice);

switch(choice)
{
   case 1:
      printf("Enter value to insert: \n");
      scanf("%d", &data);
      root=insert(root,data);
      break;
   case 2:
      printf("inorder Traversal: \n");
      inorder(root);
      printf("\n");
      break;
   case 3:
      printf("preorder Traversal: \n");
      preorder(root);
      printf("\n");
      break;
   case 4:
      printf("postorder Traversal: \n");
      postorder(root);
      printf("\n");
      break;
   case 5:
      printf("exit");
      break;
   default:
```

```
            printf("Invalid choice! Please try again.\n");
        }
    } while(choice!=5);
     freeTree(root);
     return 0;
}
```

6.BST Countleaf

```
#include<stdio.h>
#include<stdlib.h>
struct Node
{
   int val;
   struct Node *left;
   struct Node *right;
};
 struct Node* newNode(int v)
 {
   struct Node* n =(struct Node*)malloc(sizeof(struct Node));
   n->val = v;
   n->left = n->right = NULL;
   return n;
}
struct Node* insert(struct Node* node, int v)
{
   if(node == NULL) return newNode(v);
   if (v < node->val)
      node-> left = insert(node->left, v);
   else
      node-> right = insert(node->right, v);
   return node;
```

```
}
int countLeaf(struct Node* root)
{
   if(root == NULL)
     return 0;
   if (root->left == NULL && root->right == NULL)
     return 1;
   return countLeaf(root->left) + countLeaf(root->right);
}
int countNodes(struct Node* root)
{
   if (root == NULL)
     return 0;
   return countNodes(root->left) + countNodes(root->right) + 1;
}
int main()
{
   struct Node* root = NULL;
   root = insert(root, 50);
   root = insert(root, 30);
   root = insert(root, 20);
   root = insert(root, 40);
   root = insert(root, 70);
   root = insert(root, 60);
   root = insert(root, 80);

   int totalNodes = countNodes(root);
   int totalLeafNodes = countLeaf(root);
   printf("Total nodes in the BST: %d\n", totalNodes);
   printf("Total leaf nodes in the BST: %d\n", totalLeafNodes);
   return 0;
```

```
}
```

7.BST CountNode/CountHeight

```
#include <stdio.h>
#include <stdlib.h>
struct Node
{
   int val;
   struct Node *left;
   struct Node *right;
};
struct Node * newNode(int v)
{
   struct Node * n= (struct Node *)malloc (sizeof(struct Node));
   n->val = v ;
   n->left = n->right = NULL;
   return n;
}
struct Node* insert(struct Node* node, int v)
{
   if(node == NULL) return newNode(v);
   if (v < node->val)
     node-> left = insert(node->left, v);
   else
     node-> right = insert(node->right, v);
   return node;
}
void printlevel(struct Node* root, int level)
{
   if(root == NULL)
   return ;
```

```
    if(level ==1)
    {
        printf("%d", root->val);
    }
    else
    {
      printlevel(root->left, level - 1);
      printlevel(root->right, level - 1);
    }
}
int height(struct Node* node)
{
    if(node == NULL)
    return 0;
    int leftHeight = height(node->left);
    int rightHeight = height(node->right);
    return (leftHeight > rightHeight ? leftHeight : rightHeight) + 1;
}
int countleafNodes(struct Node* node)
{
    if (node == NULL)
    return 0;
    if(node->left == NULL && node->right == NULL)
    return 1;
    return countleafNodes(node->left) + countleafNodes(node->right);
}
void printlevels(struct Node* root)
{
    int h = height(root);
    printf("Total levels: %d\n",h);
    for(int i = 1; i<= h; i++)
```

```
    {
        printf("Level %d: ", i);
        printLevel(root, i);
        printf("\n");
    }
}
int main()
{
    struct Node* root = NULL;
    int vals[] = {50, 30, 20, 40, 70, 60, 88};
    int n = sizeof(vals) / sizeof(vals[0]);
    for(int i = 0; i < n; i ++);
    printLevel(root);
    int leafCount = countLeafNodes(root);
    printf("Total leaf nodes: %d\n", leafCount);
    return 0;
}
```

8.DFS

```
#include <stdio.h>
#include <stdlib.h>

#define MAX_VERTICES 100

void dfs(int graph[MAX_VERTICES][MAX_VERTICES],int visited[], int vertex, int vertices)
 {
  visited[vertex] = 1;
  printf("%d", vertex);

   for(int i = 0; i < vertices; i++)
    {
```

```
      if (graph[vertex][1] == 1 && !visited[i])
       {
        dfs(graph, visited, i, vertices);
       }
     }
   }

int main()
   {
   int vertices, edges;
   int graph[MAX_VERTICES][MAX_VERTICES] = {0};
   int visited[MAX_VERTICES] = {0};

   printf("Enter the number of vertices: ");
   scanf("%d", &vertices);
   printf("Enter the number of edges: ");
   scanf("%d", &edges);

   printf("Enter the edges (u v) where u and v are vertex indices (0 to %d):\n",vertices - 1);
   for(int i =0; i < edges; i++)
   {
   int u,v;
   scanf("%d %d", &u, &v);
   graph[u][v] = 1;
   graph[v][u] = 1;
   }

   printf("DFS Traversal: ");
   dfs(graph, visited, 0, vertices);
   printf("\n");
```

```
  return 0;
 }
```

9.BFS

```
#include<stdio.h>
#include<stdlib.h>
#define MAX 10
int graph[MAX][MAX] = {0};
int visited[MAX] = {0};
void bfs(int start, int n)
   {
      int queue[MAX] ,front = -1, rear = -1;
      queue[++rear] = start;
      visited[start] = 1;
      printf("BFS traversal ");
      while(front != rear)
        {
           int vertex = queue[++front];
           printf("%d",vertex);
           for(int i = 0; i< n ; i++)
              {
                 if(graph[vertex][i] == 1 && !visited[i])
                   {
                     queue[++rear] = i;
                     visited[i] = 1;
                   }
              }

        }
   }
```

```
int main()
  {
     int n,e,x,y;
     printf("enter number of vertices :");
     scanf("%d",&n);
     printf("enter the number of edges :");
     scanf("%d", &e);
     printf("enter the edges(start vertex,end vertex):\n");
     for(int i = 0; i<e; i++)
       {
          scanf("%d %d",&x,&y);
          graph[x][y] = 1;
          graph[y][x] = 1;
       }
      bfs(0,n);
      return 0;
  }
```

10.Kruskal's Algorithm

```
#include <stdio.h>
#include <stdlib.h>

#define MAX 100

typedef struct
{
   int s, d, w;
}Edge;

typedef struct
{
```

```
    int p;
    int r;
}Subset;

int cmp(const void *a, const void *b)
{
    return ((Edge *)a)->w > ((Edge *)b)->w;
}
int find(Subset s[], int i)
{
    if (s[i].p != i)
    {
        s[i].p = find(s, s[i].p);
    }
    return s[i].p;
}
void unionSubset(Subset s[], int x, int y)
{
    int xroot = find(s, x);
    int yroot = find(s, y);
    if (s[xroot].r < s[yroot].r)
    {
        s[xroot].p = yroot;
    }else if (s[xroot].r > s[yroot].r)
    {
        s[yroot].p = xroot;
    }else
    {
        s[yroot].p = xroot;
        s[xroot].r++;
    }
```

```
}
void kruskal(Edge e[], int numE, int numV)
{
   Edge result[MAX];
   int eCount = 0;
   int i = 0;
   qsort(e, numE, sizeof(e[0]), cmp);
   Subset* s = (Subset *)malloc(numV* sizeof(Subset));
   for(int v=0; v< numV; v++)
   {
      s[v].p = v;
      s[v].r = 0;
   }
   while(eCount < numV - 1 && i < numE)
   {
      Edge nextEdge = e[i++];
      int x = find(s, nextEdge.s);
      int y = find(s, nextEdge.d);
      if(x != y)
      {
         result[eCount++] = nextEdge;
         unionSubset(s, x, y);
      }
   }
   printf("Edge \tWeight\n");
   for (i = 0; i< eCount; i++)
   {
      printf("%d - %d \t%d \n", result[i].s, result[i].d, result[i].w);
   }
   free(s);
}
```

```
int main()
{
    Edge edges[] ={
        {0, 1, 10},
        {0, 2, 6},
        {0, 3, 5},
        {1, 3, 15},
        {2, 3, 4}
    };
    int numE = sizeof(edges) / sizeof(edges[0]);
    int numV = 4;
    kruskal(edges, numE, numV);
    return 0;
}
```

11.HeapSort

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
void heapify(int arr[], int n, int i)
{
    int largest = i;
    int left = 2 * i + 1;
    int right = 2 * i + 2;
    if(left < n && arr[left] > arr[largest])
    {
        largest = left;
    }
    if(right < n && arr[right] > arr[largest])
    {
        largest = right;
```

```
    }
    if (largest !=i)
    {
        int temp = arr[i];
        arr[i] = arr[largest];
        arr[largest] = temp;
        heapify(arr, n, largest);
    }
}
void heapSort(int arr[], int n)
{
    for (int i = n/2-1; i >= 0; i--)
    {
        heapify(arr, n, i);
    }
    for(int i = n - 1; i >= 0; i--)
    {
        int temp = arr[0];
        arr[0] = arr[i];
        arr[i] = temp;
        heapify(arr, i, 0);
    }
}
void printArray(int arr[], int n)
{
    for(int i = 0; i < n; i++)
    {
        printf("%d\t", arr[i]);
    }
    printf("\n");
}
```

```
int main()
{
   int n;
   srand(time(NULL));
   printf("Enter the number of elements: ");
   scanf("%d", &n);
   int arr[n];
   for(int i = 0; i < n; i++)
   {
      arr[i] = rand() % 100;
   }
   printf("Original array: \n");
   printArray(arr, n);
   heapSort(arr, n);
   printf("Sorted array: \n");
   printArray(arr, n);
   return 0;
}
```

12.Topology

```
#include <stdio.h>
#include <stdlib.h>

#define MAX 100

void topologicalsort(int graph[MAX][MAX], int n)
 {
  int in_degree[MAX] = {0};
  int sorted[MAX];
  int index = 0;
```

```
for (int i = 0;i < n; i++)
  {
    for (int j = 0;j < n; j++)
    {
    if (graph[i][j] == 1)
     {
       in_degree[j]++;
     }
    }
  }

for (int count = 0; count < n; count++)
 {
  int found = 0;

  for (int i =0; i < n; i++)
   {
     if(in_degree[i] == 0)
      {
        sorted[index++] = i;
        in_degree[i] = -1;

        for(int j = 0; j < n; j++)
        {
          if(graph[i][j] == 1)
          {
            in_degree[j]--;
          }
        }
      found = 1;
      break;
```

```
    }
   }
   if (!found)
   {
    printf("Graph has a cycle, topological sorting not possible.\n");
    return;
    }
   }
   printf("Topological sort: ");
   for (int i = 0;i < n; i++)
    {
     printf("%d", sorted[i]);
    }
   printf("\n");
  }

int main()
{
  int n;
  int graph[MAX][MAX];

   printf("Enter the number of vertices (max 100): ");
   scanf("%d", &n);

   if (n > MAX)
   {
    printf("Number of vertices exceeds the maximum limit.\n");
   }

   printf("Enter the adjacency matrix (0 or 1):\n");
     for (int i =0; i < n; i++)
```

```
    {
     for (int j =0; j < n; j++)
     {
      scanf("%d", &graph[i][j]);
     }
    }
  topologicalsort(graph, n);

  return 0;
}
```