DS LAB PROGRAMS(20CS37)

1) **Design, Develop and Implement a Program in C for the following operations on Strings**

   a. Read a main String (STR), a Pattern String (PAT) and a Replace String (REP)
   b. Perform Pattern Matching Operation: Find and Replace all occurrences of PAT in STR with REP if PAT exists in STR. Report suitable messages in case PAT does not exist in STR.
   c. Pattern Matching Algorithm: Brute Force
   d. Support the program with functions for each of the above operations. Don't use Built-in functions
   e. Check the following test cases.
   Test Case 1: STR = "VVCE MYSURU", PAT=" MYSURU", REP=" KARNATAKA", OUTPUT=" VVCE KARNATAKA"
   Test Case 2: STR = "COMPUTER SCIENCE", PAT=" COMPUTER", REP=" BASIC", OUTPUT=" BASIC SCIENCE"

```
#include<stdio.h>
#include<stdlib.h>
char str[100], pat[50], rep[50], ans[100];
int i, j, c, m, k, flag=0;
void stringmatch()
{
        i = m = c = j = 0;
        while(str[c] != '\0')
        {
                if(str[m] == pat[i])
                {
                i++; m++;
                        if(pat[i] == '\0')
                        {
                        flag = 1;
                        for(k = 0; rep[k] != '\0'; k++, j++)
                        ans[j] = rep[k];
                        i = 0;
                        c = m;
                        }
                }
                else
                {
                ans[j] = str[c];
                j++;
                c++;
                m=c;
                }
        }
ans[j] = '\0';
}
void main()
{
        printf("Enter a main string \n");
        gets(str);
        printf("Enter a pattern string \n");
        gets(pat);
        printf("Enter a replace string \n");
        gets(rep);
        stringmatch();
        if(flag == 1)
        printf("The resultant string is\n %s" , ans);
        else
        printf("Pattern string NOT found\n");
}
```

2) Design, Develop and Implement a Program in C for the following operations on expression.

a. Read infix expression String (INFIX)
b. Convert the infix expression (INFIX) to a postfix expression using stacks.
c. Evaluate the postfix expression using stacks.
d. Check the following test cases.
Test Case 1: Infix = “(1+ (2-3) *4)”, Postfix=”123-4*+”, Result = -3
Test Case 2: Infix = “4/2-2+3*3-4*2”, Postfix=”42/233*42*-+-”, Result = -1

Note: Program should support for both parenthesized and free parenthesized expressions with the operators: +, -, *, /, % (Remainder), ^ (Power) and alphanumeric operands.

```
#include <stdio.h>
#include <ctype.h>
#include <math.h>
char stack[100];
int top = -1;
void push(char x)
{
        stack[++top] = x;
}
char pop()
{
        if(top == -1)
        return -1;
        else
        return stack[top--];
}
int priority(char x)
{
        if(x == '(')
        return 0;
        if(x == '+' || x == '-')
        return 1;
        if(x == '*' || x == '/' || x=='%')
        return 2;
        if(x=='^')
        return 3;
return 0;
}
void main()
{
        char exp[20];
        char *e, x;
        printf("enter the expression : ");
        scanf("%s", exp);
        printf("\n");
```

```
    e = exp;
    while (*e != '\0')
    {
        if(isalnum(*e))
        printf("%c", *e);
        else if (*e == '(')
        push(*e);
        else if(*e == ')')
        {
        while ((x = pop()) != '(')
        printf("%c", x);
        }
        else{
        while(priority(stack[top]) >= priority(*e))
        printf("%c", pop());
        push(*e);
        }
    e++;
    }
    while(top != -1)
    {
    printf("%c", pop());
    }
char postfix[20];
char *p;
int n1,n2,n3,num;
printf("\nEnter the result to calculate :: ");
scanf("%s",postfix);
p = postfix;
while(*p != '\0')
{
    if(isdigit(*p))
    {
        num = *p - 48;
        push(num);
    }
    else
    {
    n1 = pop();
    n2 = pop();
    switch(*p)
    {
      case '+':
      {
         n3 = n2 + n1;
         break;
      }
      case '-':
      {
        n3 = n2 - n1;
        break;
      }
      case '*':
      {
        n3 = n2 * n1;
        break;
      }
      case '/':
      {
        n3 = n2 / n1;
        break;
      }
      case '^':
      {
        n3 = pow(n2,n1);
        break;
      }
      case '%':
      {
        n3 = n2%n1;
        break;
      }
      }
      push(n3);
    }
    p++;
  }
  printf("\nThe result of the converted postfix =  %d",pop());
}
```

3) Design, Develop and implement menu driven program to simulate processing of batch jobs by a computer system. The scheduling of these jobs should be handled using a priority queue.

Note: The Program should allow users to add or remove items from the queue and it should also display current status i.e. the total number of items in the queue.

```
#include <stdio.h>
#include<stdlib.h>
#define MAX 5
int front=-1,rear=-1;
typedef struct process
{
        int pid;
        int pr;
        int bt;
}job;
job pjob[MAX];
void insert()
{
        int pid,pr,bt;
        if(rear==MAX-1)
        {
                printf("Overflow");
        }
        else
        {
                printf("Enter PID, PR AND BT: ");
                scanf("%d %d %d",&pid,&pr,&bt);
                if(rear==-1)
                {
                        rear++;
                        front++;
                }
                else
                {
                        rear++;
                }
                pjob[rear].pid=pid;
                pjob[rear].pr=pr;
                pjob[rear].bt=bt;
        }
}
void delete()
{
        int i, pos=0,max=0;
        if(front==-1)
        {
                printf("Underflow\n");
        }
```

```
    else
    {
        if(front==rear)
        {
            front=-1;
            rear=-1;
        }
        else
        {
            for(i=front;i<=rear;i++)
            {
                if(pjob[i].pr>max)
                {
                    max=pjob[i].pr;
                    pos=i;
                }
            }
            for(i=pos;i<=rear;i++)
            {
                pjob[i].pid=pjob[i+1].pid;
                pjob[i].pr=pjob[i+1].pr;
                pjob[i].bt=pjob[i+1].bt;
            }
        rear--;
        }
    }
}
void display()
{
    if(front==-1)
    {
        printf("Queue is Empty\n");
    }
    else
    {
        for(int i=front;i<=rear;i++)
        {
            printf("PID\t PR\t BT\n");
            printf("%d\t %d\t %d\n",pjob[i].pid,pjob[i].pr,pjob[i].bt);
        }
    }
}
void main()
{
    int ch;
    while(1)
    {
        printf("\n1.Insert\t 2.Display\t 3.Delete\t 4.Exit\n");
        printf("\nEnter your choice: ");
        scanf("%d", &ch);
        switch(ch)
        {
            case 1: insert();
            break;
            case 2: display();
            break;
            case 3: delete();
            break;
            case 4: exit(0);
            break;
            default: printf("\nInvalid choice:\n");
            break;
        }
    }
}
```

4) Design, Develop and implement c program using singly linked list for the following scenario

a. There are two linked list A and B containing the following data: A: 3,7,10,15,16,09,22,17,32 and B: 16,02,09,13,37,08,10,01,28

b. Create a linked list C that contains only those elements that are common in linked list A and B

c. Create a linked list D which contains all elements of A as well as B ensures that there is no repetition of elements.

```
#include <stdio.h>
#include <stdlib.h>
typedef struct node
{
 int data;
 struct node *link;
}NODE;
NODE *LLone, *LLtwo, *unionLL, *interLL;
NODE* insert(NODE **first, int num)
{
   NODE* newNode = (NODE*) malloc(sizeof(NODE));
   newNode->data  = num;
   newNode->link = *first;
   *first = newNode;
   return *first;
}
int search(NODE *first, int num)
{
   while (first != NULL) {
      if (first->data == num)
     {
         return 1;
      }
       first= first->link;
   }
   return 0;
}
NODE* findunion(NODE *LLone, NODE *LLtwo)
{
   unionLL = NULL;
   NODE *temp=LLone;
   while(temp != NULL){
      insert(&unionLL, temp->data);
      temp = temp->link;
   }
   while(LLtwo != NULL){
      if(!search(LLone, LLtwo->data)){
         insert(&unionLL, LLtwo->data);
      }
      LLtwo = LLtwo->link;
   }
   return unionLL;
}
NODE* intersection(NODE *LLone, NODE *LLtwo)
{
   interLL = NULL;
   while(LLone != NULL){
      if(search(LLtwo, LLone->data))
     {
         insert(&interLL, LLone->data);
      }
      LLone = LLone->link;
   }
   return interLL;
}
void printList(NODE *cur)
{
  while (cur!= NULL) {
    printf("-->%d", cur->data);
    cur = cur->link;
  }
}
void main()
{
   int i, LLonecount, LLtwocount, temp;
   printf("\n Enter number of nodes in first Linked List: ");
   scanf("%d", &LLonecount);
   printf("\n Enter data of first linked list: ");
   for(i=0; i<LLonecount; i++)
   {
      scanf("%d", &temp);
      insert(&LLone, temp);
   }
   printList(LLone);
   printf("\n Enter number of nodes in second Linked List: ");
   scanf("%d", &LLtwocount);
   printf("\n Enter data of second linked list: ");
   for(i=0; i<LLtwocount; i++)
   {
      scanf("%d", &temp);
      insert(&LLtwo, temp);
   }
   printList(LLtwo);
   findunion(LLone, LLtwo);
   intersection(LLone, LLtwo);
   printf("\nUnion Linked List\n");
   printList(unionLL);
   printf("\nIntersection Linked List\n");
   printList(interLL);
}
```

5) Design, Develop and implement C program for the following operations on doubly linked list.

a. Create doubly linked list of N nodes with integer data by adding each node at the front.
b. Delete the node of a given data if it is found, otherwise display appropriate message.
c. Insert a node to the left of the node whose key value is read as input.
d. Display the contents of the list.

```
#include <stdio.h>
#include<stdlib.h>
typedef struct student
{
  int data;
  struct student *next, *prev;
}NODE;

NODE* getnode( )
{
  NODE *x;
  x=(NODE*)malloc(sizeof(NODE));
  printf("\n Enter Data of Node to be Inserted: ");
  scanf("%d",&x->data);
  x->next=x->prev=NULL;
 return x;
}
NODE* insert_front(NODE* first)
{
        NODE *temp;
        if(first==NULL)
        {
                temp=getnode();
                first=temp;
        }
        else
        {
                temp=getnode();
                temp->next=first;
                first->prev=temp;
                first=temp;
        }
        return first;
}
NODE* insert_left(NODE* first)
{
  NODE *temp,*cur,*pre;
  int data;
  if(first==NULL)
  {
        temp=getnode();
        first=temp;
  }
 else
 {
        printf("Enter the node data to which left part new node to be inserted: ");
        scanf("%d",&data);
         temp=getnode();
        cur=first;
        while(cur->data!=data)
        {
                 pre=cur;
                cur=cur->next;
         }
         pre->next=temp;
         temp->prev=pre;
         temp->next=cur;
        cur->prev=temp;
 }
 return first;
}

NODE* delete_node(NODE* first)
{
 NODE *cur;
 int data;
 cur=first;
 printf("Enter the data of the NODE to be deleted:  ");
 scanf("%d",&data);
 if(first==NULL)
 {
        printf("\n List is empty\n");
 }
 else if(first->data==data)
 {
         first=first->next;
         free(cur);
 }
 else
 {
        while(cur!=NULL)
        {
                if(cur->data==data)
                break;
                cur=cur->next;
        }
        if(cur!=NULL)
        {
                if(cur->next!=NULL)
                {
                        (cur->next)->prev=cur->prev;
                        (cur->prev)->next=cur->next;
                        free(cur);
                }
                else
                {
                        (cur->prev)->next=NULL;
                        free(cur);
                }
        }
        else
        {
                printf("No such node is present in the list\n");
        }
 }
 return first;
}

NODE* display(NODE* first)
{
        NODE *cur;
        if(first == NULL)
                printf("No nodes present\n");
        else
        {
                cur=first;
                while(cur!=NULL)
                {
                printf("-->%d", cur->data);
                cur = cur->next;
                }
        }
        return first;
}
int main()
{
 NODE *first;
 first=NULL;
 int ch;
 while(1)
 {
  printf("\n1.InsertFront\t 2. InsertLeft\t 3.Delete\t 4.Display\t 5.exit\n");
  printf("Enter Your Choice: ");
  scanf("%d",&ch);
  switch(ch)
  {
   case 1:first=insert_front(first);
   break;
   case 2:first=insert_left(first);
   break;
   case 3:first=delete_node(first);
   break;
   case 4:first=display(first);
```

```
      break;
      case 5:exit(0);
      break;
      default: printf("\n Invalid choice\n");
      break;
    }
  }
  return 0;
}
```

6) Design, Develop and Implement a menu driven Program in C for the following operations on Binary Search Tree (BST) of Integers.

a. Create a BST of N Integers: 6, 9, 5, 2, 8, 15, 24, 14, 7, 8, 5, 2
b. Traverse the BST in In-order, preorder, post-Order
c. Search the BST for a given element (KEY) and report the appropriate message
d. Display the height of binary trees
e. Exit

```
#include<stdio.h>
#include<stdlib.h>
typedef struct node
{
 int item;
 struct node *llink, *rlink;
}NODE;

NODE* getnode()
{
 NODE* x;
 x=(NODE*)malloc(sizeof(NODE));
 scanf("%d",&x->item);
 x->llink=x->rlink=NULL;
 return x;
}

NODE* insert(NODE* root)
{
 NODE *temp,*cur,*prev;
 temp=getnode();
 if(root==NULL)
 {
        root=temp;
 }
 else
 {
        prev=NULL;
         cur=root;
        while(cur!=NULL)
        {
                prev=cur;
                if(temp->item<cur->item)
                cur=cur->llink;
                else
                cur=cur->rlink;
        }
        if(temp->item<prev->item)
        prev->llink=temp;
        else
         prev->rlink=temp;
 }
return root;
}

void search(NODE *root)
{
 int item;
 NODE *cur;
 cur=root;
 if(root==NULL)
 {
        printf("Tree is empty\n");
 }
 else
 {
        printf("Enter the item to be searched: ");
        scanf("%d",&item);
        while(cur!=NULL)
        {
                if(cur->item==item)
                 break;
                if(cur->item<item)
                cur=cur->rlink;
                else
                cur=cur->llink;
         }
         if(cur!=NULL)
         {
                printf("Item found\n");
        }
        else
        {
                printf("Item Not found");
        }
 }
}

void preorder(NODE *root)
{
 if(root==NULL) return;
 printf("%d\t",root->item);
 preorder(root->llink);
 preorder(root->rlink);
}

void postorder(NODE *root)
{
 if(root==NULL) return;
 postorder(root->llink);
 postorder(root->rlink);
 printf("%d\t",root->item);
}

void inorder(NODE *root)
{
 if(root==NULL) return;
 inorder(root->llink);
 printf("%d\t",root->item);
 inorder(root->rlink);
}
int find_height(NODE *root)
{
  if (root==NULL)
  {
   return -1;
   }
  else
  {
    int lheight = find_height(root->llink);
    int rheight = find_height(root->rlink);
    if (lheight > rheight)
       return(lheight + 1);
    else
       return(rheight + 1);
  }
}
int main()
{
 int ch,i,n,ht;
 NODE *root=NULL;
 while(1)
 {
  printf("\n 1.Create\t 2.Traverse\t 3.Search\t 4.Height\t 5.Exit\n");
  printf("Enter your choice: ");
  scanf("%d",&ch);
   switch(ch)
   {
    case 1:printf("Enter the number of nodes to be inserted: ");
          scanf("%d",&n);
          printf("Enter the tree nodes\n");
          for(i=0;i<n;i++)
          {
           root=insert(root);
          }
          break;
    case 2:printf("\n Preorder Traversal: ");
          preorder(root);
          printf("\n Inorder Traversal: ");
          inorder(root);
          printf("\n Postorder Traversal: ");
          postorder(root);
          break;
      case 3:search(root);
          break;
      case 4:ht=find_height(root);
          printf("\n Height of the tree = %d\n",ht);
          break;
      case 5:exit(0);
      default:printf("\n Invalid Choice\n");
       break;
 }
 }
 return 0;
}
```

Program-7: Design, develop a program in C to implement AVL tree operations.

```
#include<stdio.h>
typedef struct node
{
int data;
struct node *left,*right;
int ht;
}NODE;

int height(NODE *T)
{
   if (T==NULL)
   {
     return -1;
    }
   else
   {
      int lheight = height(T->left);
      int rheight = height(T->right);
      if (lheight > rheight)
           return(lheight + 1);
      else
           return(rheight + 1);
   }
}

int BF(NODE *T)
{
  int lh,rh;
  if(T==NULL)
    return 0;
  if(T->left==NULL)
    lh=0;
  else
    lh=1+T->left->ht;
  if(T->right==NULL)
    rh=0;
  else
    rh=1+T->right->ht;
return(lh-rh);
}

NODE * rotateright(NODE *x)
{
  NODE *y;
  y=x->left;
  x->left=y->right;
  y->right=x;
```

```
 x->ht=height(x);
 y->ht=height(y);
 return y;
}

NODE * rotateleft(NODE *x)
{
 NODE *y;
 y=x->right;
 x->right=y->left;
 y->left=x;
 x->ht=height(x);
 y->ht=height(y);
 return y;
}

NODE* RR(NODE *T)
{
T=rotateleft(T);
return T;
}

NODE* LL(NODE *T)
{
T=rotateright(T);
return T;
}

NODE* LR(NODE *T)
{
T->left=rotateleft(T->left);
T=rotateright(T);
return T;
}

NODE* RL(NODE *T)
{
T->right=rotateright(T->right);
T=rotateleft(T);
return T;
}

NODE* insert(NODE *T, int x)
{
 if(T==NULL)
 {
  T=(NODE*)malloc(sizeof(NODE));
  T->data=x;
  T->left=T->right=NULL;
 }
 else
 if(x > T->data)
 {
  T->right=insert(T->right,x);
  if(BF(T)==-2)
  if(x>T->right->data)
  T=RR(T);
  else
  T=RL(T);
 }
 else
 if(x<T->data)
 {
  T->left=insert(T->left,x);
  if(BF(T)==2)
  if(x < T->left->data)
  T=LL(T);
  else
  T=LR(T);
 }
 T->ht=height(T);
return(T);
}

void inorder(NODE *T)
{
 if(T!=NULL)
 {
  inorder(T->left);
  printf("%d(Bf=%d)",T->data,BF(T));
  inorder(T->right);
 }
}

int main()
{
NODE *root=NULL;
int x,n,i,ch;
while(1)
{
printf("\n 1.Create\t 2.Insert\t 3.Display\t 4.Exit\n");
printf("\nEnter Your Choice:");
scanf("%d",&ch);
switch(ch)
{
case 1: printf("\nEnter no. of elements:");
        scanf("%d",&n);
        printf("\nEnter tree data:");
        root=NULL;
        for(i=0;i<n;i++)
        {
        scanf("%d",&x);
        root=insert(root, x);
        }
    break;
case 2: printf("\nEnter a data:");
        scanf("%d",&x);
        root=insert(root,x);
        break;
case 3: printf("\nInorder sequence:\n");
       inorder(root);
        break;
case 4:exit(0);
}
}
return 0;
}
```

8) Design, Develop a program in C to implement various operations on Red-Black Tree.

```
#include<stdio.h>
#include<stdlib.h>
typedef struct NODE{
   int key;
   char color;
   struct NODE *left, *right,*parent;
}NODE;
NODE *root = NULL;

void leftRotate(NODE *x){
   NODE *y;
   y = x->right;
   x->right = y->left;
   if( y->left != NULL)
   {
      y->left->parent = x;
   }
   y->parent = x->parent;
   if( x->parent == NULL){
      root = y;
   }
   else if((x->parent->left!=NULL) && (x->key == x->parent->left->key))
   {
      x->parent->left = y;
   }
   else x->parent->right = y;
   y->left = x; x->parent = y; return;
}

void rightRotate(NODE *y){
   NODE *x;
   x = y->left;
   y->left = x->right;
   if ( x->right != NULL)
   {
      x->right->parent = y;
   }
   x->parent = y->parent;
   if( y->parent == NULL)
   {
      root = x;
   }
   else if((y->parent->left!=NULL)&& (y->key == y->parent->left->key))
   {
      y->parent->left = x;
   }
```

```
  else
  y->parent->right = x;
  x->right = y; y->parent = x;
  return;
}
void colorinsert(NODE *z){
  NODE *y=NULL;
  while ((z->parent != NULL) && (z->parent->color == 'r'))
  {
    if ((z->parent->parent->left != NULL) && (z->parent->key == z->parent->parent->left-
>key))
    {
      if(z->parent->parent->right!=NULL)
        y = z->parent->parent->right;
      if ((y!=NULL) && (y->color == 'r'))
      {
        z->parent->color = 'b';
        y->color = 'b';
        z->parent->parent->color = 'r';
        if(z->parent->parent!=NULL)
          z = z->parent->parent;
      }
      else
      {
        if ((z->parent->right != NULL) && (z->key == z->parent->right->key))
        {
          z = z->parent;
          leftRotate(z);
        }
        z->parent->color = 'b';
        z->parent->parent->color = 'r';
        rightRotate(z->parent->parent);
      }
    }
    else
    {
      if(z->parent->parent->left!=NULL)
        y = z->parent->parent->left;
      if ((y!=NULL) && (y->color == 'r'))
      {
        z->parent->color = 'b';
        y->color = 'b';
        z->parent->parent->color = 'r';
        if(z->parent->parent!=NULL)
          z = z->parent->parent;
      }
      else
      {
        if ((z->parent->left != NULL) && (z->key == z->parent->left->key))
```

```
            {
               z = z->parent;
               rightRotate(z);
            }
            z->parent->color = 'b';
            z->parent->parent->color = 'r';
            leftRotate(z->parent->parent);
         }
      }
   }
   root->color = 'b';
}

void inorder(NODE* root){
   NODE* temp = root;
   if (temp != NULL)
   {
      inorder(temp->left);
      printf(" %d-%c ",temp->key,temp->color);
      inorder(temp->right);
   }
   return;
}

void insert(int val){
   NODE *cur, *prev;
   NODE *z = (NODE*)malloc(sizeof(NODE));
   z->key = val;
   z->left = NULL;
   z->right = NULL;
   z->color = 'r';
   cur=root;
   if ( root == NULL )
   {
      root = z;
      root->color = 'b';
      return;
   }
   while ( cur != NULL)
   {
      prev = cur;
      if ( z->key < cur->key)
      {
         cur = cur->left;
      }
      else
         cur = cur->right;
   }
   z->parent = prev;
```

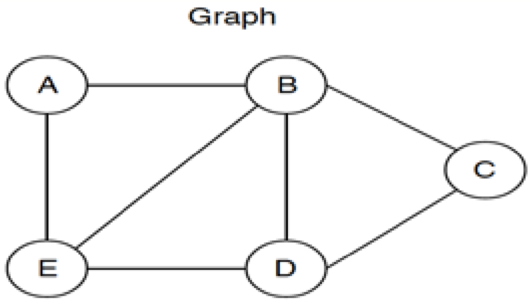
```
    if ( prev == NULL)
    {
        root = z;
    }
    else if( z->key < prev->key )
    {
        prev->left = z;
    }
    else{
        prev->right = z;
    }
    colorinsert(z);
    return;
}

int main()
{
    int choice, val;
    while(1)
    {
    printf("\nRed Black Tree Menu - \nEnter your choice :\n1:Insert\n2:Traversal
\n3:Exit\n");
        scanf("%d",&choice);
        switch(choice)
        {
            case 1:printf("Enter the integer you want to add : ");
                scanf("%d",&val);
                insert(val);
            break;
            case 2:inorder(root);
            break;
            case 3: exit(0);
            default: printf("\nInvalid Choice\n");
        }
        }
    return 0;
}
```

9) Design, Develop and Implement a Program in C for the following operations on Graph (G) of Cities

a. Create a Graph of N cities using Adjacency Matrix.

b. Print all the nodes reachable from a given starting node in a digraph using the DFS / BFS method



DFS:

```
#include<stdio.h>
int stack[10];
int top=-1;
int adj[10][10];
int vis[10]={0};
void main()
{
        int n, s, u, v, i, j;
        int found=0;
        printf("\n Enter the number of vertex:");
        scanf("%d",&n);
        printf("\n Enter the adj matrix:\n");
        for(i=0;i<n;i++)
        {
                for(j=0;j<n;j++)
                {
                        scanf("%d",&adj[i][j]);
                }
        }
        printf("\n Enter the source vertex:");
        scanf("%d",&s);
        stack[++top]=s;
        vis[s]=1;
        printf("source %d:",s);
        while(top!=-1)
        {
                found=0;
                u=stack[top];
                for(v=0;v<n && found==0;v++)
                {
                        if(adj[u][v]==1 && vis[v]==0)
                        {
                                printf("->%d",v);
                                stack[++top]=v;
                                vis[v]=1;
                                found=1;
                }
            }
            if(found==0)
            {
                top--;
            }
        }
}
```

BFS:

```
#include<stdio.h>
int q[10];
int r=-1, f=0;
int adj[10][10];
int vis[10]={0};
void main()
{
        int n, i, j, s, v, u;
        printf("\n Enter the number of vertex:");
        scanf("%d",&n);
        printf("\n Enter the Adj matrix:\n ");
        for(i=0;i<n;i++)
        {
            for(j=0;j<n;j++)
            {
                scanf("%d",&adj[i][j]);
            }
        }
        printf("\n Enter the source vertex:");
        scanf("%d",&s);
        q[++r]=s;
        vis[s]=1;
        printf("%d: ",s);
        while(f<=r)
        {
            u=q[f++];
            for(v=0;v<n;v++)
            {
                if(adj[u][v]==1 && vis[v]==0)
                {
                        printf("->%d",v);
                        vis[v]=1;
                        q[++r]=v;
                }
            }
        }
}
```

10) Given a File of N employee records with a set K of Keys (4-digit) which uniquely determine the records in file F.

a. Assume that file F is maintained in memory by a Hash Table (HT) of M memory locations with L as the
set of memory addresses (2-digit) of locations in HT.

b. Let the keys in K and addresses in L are Integers. Design and develop a Program in C that uses Hash
Function H: K%L as I (remainder method), and implement hashing techniques to map a given key K to
the address space L.

c. Resolve the collision (if any) using linear probing

```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
#define SIZE 10
typedef struct
{
	int id;
	char name[20];
}EMPLOYEE;
EMPLOYEE e[SIZE];

void initialize_table()
{
	for(int i=0; i<SIZE; i++)
	{
		e[i].id=0;
	}
}

void insert_table()
{
 int i,id,index,hvalue;
 char name[26];
 printf("Enter the employee id and name: ");
 scanf("%d %s",&id,name);
	hvalue= id % SIZE;
	for(i=0; i<SIZE; i++)
	{
		index=(hvalue+i) % SIZE;
		if(e[index].id==0)
		{
			e[index].id=id;
			strcpy(e[index].name,name);
			break;
		}
	}
 if(i==SIZE)
 {
   printf("Hash table full\n");
 }
}

void display_table()
{
        printf("H\t Id\t Name\n");
        for(int i=0; i<SIZE; i++)
        {
     printf("%d\t %d\t %s\n",i,e[i].id,e[i].name);
        }
}

void main()
{
        int ch;
        initialize_table();
        while(1)
        {
                printf("1:Insert\t 2:Display\t 3:Exit\n");
                printf("Enter the choice:");
                scanf("%d",&ch);
                switch(ch)
                {
                        case 1:insert_table( );
                        break;
                        case 2:display_table();
                        break;
                        case 3: exit(0);
                        break;
                        deafult: printf("Enter valid choice\n");
                        break;
                }
        }
}
```