ADS Lab Programs

1. Binary tree – recursive

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

typedef struct node {
    int data;
    struct node * lchild;
    struct node * rchild;
}BinTree;

BinTree *root;

BinTree *createNode() {
    BinTree* newnode;
    newnode = (BinTree*)malloc(sizeof(BinTree));
    printf("Enter data: ");
    scanf(" %d",&newnode->data);
    newnode->lchild=NULL;
    newnode->rchild=NULL;
    return newnode;
}

BinTree *create() {
    int ch;
    BinTree *t;
    t=createNode();
    puts("if you want left child press 1");
    scanf("%d",&ch);
    if(ch==1) {
        t->lchild=create();
    }
    puts("for right child press 1: ");
    scanf("%d",&ch);
    if(ch==1) {
        t->rchild=create();
    }
    return t;
}

void inorder(BinTree *t) {
    if(t) {
        inorder(t->lchild);
        printf("%d ",t->data);
        inorder(t->rchild);
    }
}

void preorder(BinTree *t) {
    if(t) {
        printf("%d",t->data);
        preorder(t->lchild);
        preorder(t->rchild);
    }
}

void postorder(BinTree *t) {
    if(t) {
        postorder(t->lchild);
        postorder(t->rchild);
        printf("%d",t->data);
    }
}

int main() {
    BinTree *root;
    int choice;
    while(1) {
        puts("1.Create \n 2.Inorder \n 3.preorder \n 4.postorder \n 5.exit");
        printf("Enter your choice: ");
        scanf("%d",&choice);
        switch(choice) {
            case 1:
                root=create();
                break;
            case 2:
                printf("Inorder traversal\n");
                inorder(root);
                break;
            case 3:
                printf("Preorder traversal\n");
                preorder(root);
                break;
            case 4:
                printf("Post order traversal\n");
                postorder(root);
                break;
            case 5:
                exit(0);
        }
    }
}
```

2. Binary Tree – Non Recursive

```c
//binary tree non recursive
#include <stdio.h>
#include <stdlib.h>

#define SIZE 20

typedef struct node {
    int data;
    struct node* lchild;
    struct node* rchild;
} BinTree;

BinTree* Stack[SIZE];
int top = -1;

BinTree* createNode() {
    BinTree* newnode;
    newnode = (BinTree*)malloc(sizeof(BinTree));
    printf("Enter data: ");
    scanf(" %d", &newnode->data);
    newnode->lchild = NULL;
    newnode->rchild = NULL;
    return newnode;
}

BinTree* create() {
    int ch;
    BinTree* t;
    t = createNode();
    puts("If you want left child press 1");
    scanf("%d", &ch);
    if (ch == 1) {
        t->lchild = create();
    }
    puts("For right child press 1: ");
    scanf("%d", &ch);
    if (ch == 1) {
        t->rchild = create();
    }
    return t;
}

void push(BinTree* t) {
    if (top == SIZE - 1) {
        printf("Overflow\n");
    } else {
        top = top + 1;
        Stack[top] = t;
    }
}

BinTree* pop() {
    if (top == -1) {
        printf("Underflow\n");
        return NULL;
    } else {
        return Stack[top--];
    }
}

void inorder(BinTree* t) {
    while (1) {
        if (t != NULL) {
            push(t);
            t = t->lchild;
        } else if (top != -1) {
            t = pop();
            printf("%d ", t->data);
            t = t->rchild;
        } else {
            break;
        }
    }
}

void preorder(BinTree* t) {
    while (1) {
        if (t != NULL) {
            printf("%d ", t->data);
            push(t);
            t = t->lchild;
        } else if (top != -1) {
            t = pop();
            t = t->rchild;
        } else {
            break;
        }
    }
}

void postorder(BinTree* t) {
    BinTree* prev = NULL;
    while (t != NULL || top != -1) {
        if (t != NULL) {
            push(t);
            t = t->lchild;
        } else {
            t = Stack[top];
            if (t->rchild == NULL || t->rchild == prev) {
                printf("%d ", t->data);
                prev = pop();
                t = NULL;
            } else {
                t = t->rchild;
            }
        }
    }
}

int main() {
    BinTree* root = NULL;
    int choice;

    while (1) {
        printf("\nMenu:\n");
        printf("1. Create Binary Tree\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:
                root = create();
                break;
            case 2:
                if (root == NULL) {
                    printf("Tree is empty\n");
                } else {
                    printf("Inorder Traversal: ");
                    inorder(root);
                    printf("\n");
                }
                break;
            case 3:
                if (root == NULL) {
                    printf("Tree is empty\n");
                } else {
                    printf("Preorder Traversal: ");
                    preorder(root);
                    printf("\n");
                }
                break;
            case 4:
                if (root == NULL) {
                    printf("Tree is empty\n");
                } else {
                    printf("Postorder Traversal: ");
                    postorder(root);
                    printf("\n");
                }
                break;
            case 5:
                exit(0);
            default:
                printf("Invalid choice\n");
        }
    }

    return 0;
}
```

3. Binary Search Tree – Recursive

```c
//binary search tree recursive
#include <stdio.h>
#include <stdlib.h>

typedef struct node {
    int data;
    struct node* lchild;
    struct node* rchild;
} BinTree;

BinTree* root = NULL;

struct node* createNode(int value) {
    struct node* newNode;
    newNode = (struct node*)malloc(sizeof(struct node));
    newNode->data = value;
    newNode->lchild = NULL;
    newNode->rchild = NULL;
    return newNode;
}

struct node* insertNode(struct node* t, int value) {
    if (t == NULL) {
        t = createNode(value);
    } else {
        if (value > t->data) {
            t->rchild = insertNode(t->rchild, value);
        } else if (value < t->data) {
            t->lchild = insertNode(t->lchild, value);
        } else {
            printf("Value already exists\n");
        }
    }
    return t;
}

BinTree* findMin(BinTree* t) {
    while (t->lchild != NULL) {
        t = t->lchild;
    }
    return t;
}

void deleteNode(int item) {
    BinTree* ptr = root;
    BinTree* parent = NULL;
    BinTree* ptr1;
    int flag = 0, option;

    while (ptr != NULL && flag == 0) {
        if (item == ptr->data) {
            flag = 1;
        } else if (item < ptr->data) {
            parent = ptr;
            ptr = ptr->lchild;
        } else {
            parent = ptr;
            ptr = ptr->rchild;
        }
    }

    if (ptr == NULL) {
        printf("Element not found\n");
        return;
    } else {
        if (ptr->lchild == NULL && ptr->rchild == NULL) {
            option = 1;
        } else if (ptr->lchild != NULL && ptr->rchild != NULL) {
            option = 3;
        } else {
            option = 2;
        }
    }

    if (option == 1) {
        if (parent->lchild == ptr) {
            parent->lchild = NULL;
        } else {
            parent->rchild = NULL;
        }
        free(ptr);
    } else if (option == 2) {
        if (parent->lchild == ptr) {
            if (ptr->lchild != NULL) {
                parent->lchild = ptr->lchild;
            } else {
                parent->lchild = ptr->rchild;
            }
        } else if (parent->rchild == ptr) {
            if (ptr->lchild != NULL) {
                parent->rchild = ptr->lchild;
            } else {
                parent->rchild = ptr->rchild;
            }
        }
        free(ptr);
    } else if (option == 3) {
        ptr1 = findMin(ptr->rchild);
        int minValue = ptr1->data;
        deleteNode(minValue);
        ptr->data = minValue;
    }
}

void inorderTraversal(BinTree* t) {
    if (t != NULL) {
        inorderTraversal(t->lchild);
        printf("%d ", t->data);
        inorderTraversal(t->rchild);
    }
}

int main() {
    int choice, value;

    while (1) {
        printf("\nMenu:\n");
        printf("1. Insert Node\n");
        printf("2. Delete Node\n");
        printf("3. Inorder Traversal\n");
        printf("4. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                printf("Enter value to insert: ");
            scanf("%d", &value);
            root = insertNode(root, value);
            break;
            case 2:
                printf("Enter value to delete: ");
            scanf("%d", &value);
            deleteNode(value);
            break;
            case 3:
                printf("Inorder Traversal: ");
            inorderTraversal(root);
            printf("\n");
            break;
            case 4:
                exit(0);
            default:
                printf("Invalid choice\n");
        }
    }
}
```

4. Binary Search Tree – Non Recursive

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

typedef struct node {
    int data;
    struct node* lchild;
    struct node* rchild;
} BinTree;

BinTree* root = NULL;

struct node* createNode(int value) {
    struct node* newNode;
    newNode = (struct node*)malloc(sizeof(struct node));
    newNode->data = value;
    newNode->lchild = NULL;
    newNode->rchild = NULL;
```

```c
    return newNode;
}

struct node* insertNode(struct node* t, int value) {
    if (t == NULL) {
        t = createNode(value);
    } else {
        if (value > t->data) {
            t->rchild = insertNode(t->rchild, value);
        } else if (value < t->data) {
            t->lchild = insertNode(t->lchild, value);
        } else {
            printf("Value already exists\n");
        }
    }
    return t;
}

BinTree* findMin(BinTree* t) {
    while (t->lchild != NULL) {
        t = t->lchild;
    }
    return t;
}

void deleteNode(int item) {
    BinTree* ptr = root;
    BinTree* parent = NULL;
    BinTree* ptr1;
    int flag = 0, option;

    while (ptr != NULL && flag == 0) {
        if (item == ptr->data) {
            flag = 1;
        } else if (item < ptr->data) {
            parent = ptr;
            ptr = ptr->lchild;
        } else {
            parent = ptr;
            ptr = ptr->rchild;
        }
    }

    if (ptr == NULL) {
        printf("Element not found\n");
        return;
    } else {
        if (ptr->lchild == NULL && ptr->rchild == NULL) {
            option = 1;
        } else if (ptr->lchild != NULL && ptr->rchild != NULL) {
            option = 3;
        } else {
            option = 2;
        }
    }

    if (option == 1) {
        if (parent->lchild == ptr) {
            parent->lchild = NULL;
        } else {
            parent->rchild = NULL;
        }
        free(ptr);
    } else if (option == 2) {
        if (parent->lchild == ptr) {
            if (ptr->lchild != NULL) {
                parent->lchild = ptr->lchild;
            } else {
                parent->lchild = ptr->rchild;
            }
        } else if (parent->rchild == ptr) {
            if (ptr->lchild != NULL) {
                parent->rchild = ptr->lchild;
            } else {
                parent->rchild = ptr->rchild;
            }
        }
        free(ptr);
    } else if (option == 3) {
        ptr1 = findMin(ptr->rchild);
        int minValue = ptr1->data;
        deleteNode(minValue);
        ptr->data = minValue;
    }
}

void inorderTraversal(BinTree* t) {
    if (t != NULL) {
        inorderTraversal(t->lchild);
        printf("%d ", t->data);
        inorderTraversal(t->rchild);
    }
}

void preorderTraversal(BinTree* t) {
    if (t != NULL) {
```

```c
        printf("%d ", t->data);
        preorderTraversal(t->lchild);
        preorderTraversal(t->rchild);
    }
}

void postorderTraversal(BinTree* t) {
    if (t != NULL) {
        postorderTraversal(t->lchild);
        postorderTraversal(t->rchild);
        printf("%d ", t->data);
    }
}

int main() {
    int choice, value;

    while (1) {
        printf("\nMenu:\n");
        printf("1. Insert Node\n");
        printf("2. Delete Node\n");
        printf("3. Inorder Traversal\n");
        printf("4. Preorder Traversal\n");
        printf("5. Postorder Traversal\n");
        printf("6. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                printf("Enter value to insert: ");
                scanf("%d", &value);
                root = insertNode(root, value);
                break;
            case 2:
                printf("Enter value to delete: ");
                scanf("%d", &value);
                deleteNode(value);
                break;
            case 3:
                printf("Inorder Traversal: ");
                inorderTraversal(root);
                printf("\n");
                break;
            case 4:
                printf("Preorder Traversal: ");
                preorderTraversal(root);
                printf("\n");
                break;
            case 5:
                printf("Postorder Traversal: ");
                postorderTraversal(root);
                printf("\n");
                break;
            case 6:
                exit(0);
            default:
                printf("Invalid choice\n");
        }
    }
}
```

5. AVL Tree Construction

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

typedef struct node {
    int data;
    struct node* lchild;
    struct node* rchild;
    int height;
} AVLTree;

AVLTree* root = NULL;

int height(AVLTree* N) {
    if (N == NULL)
        return 0;
    return N->height;
}

AVLTree* createNode(int value) {
    AVLTree* newNode = (AVLTree*)malloc(sizeof(AVLTree));
    newNode->data = value;
    newNode->lchild = NULL;
    newNode->rchild = NULL;
    newNode->height = 1;
    return newNode;
}

AVLTree* rightRotate(AVLTree* y) {
    AVLTree* x = y->lchild;
    AVLTree* T2 = x->rchild;

    x->rchild = y;
    y->lchild = T2;

    y->height = max(height(y->lchild), height(y->rchild)) + 1;
    x->height = max(height(x->lchild), height(x->rchild)) + 1;

    return x;
}

AVLTree* leftRotate(AVLTree* x) {
    AVLTree* y = x->rchild;
    AVLTree* T2 = y->lchild;

    y->lchild = x;
    x->rchild = T2;

    x->height = max(height(x->lchild), height(x->rchild)) + 1;
    y->height = max(height(y->lchild), height(y->rchild)) + 1;

    return y;
}

int getBalance(AVLTree* N) {
    if (N == NULL)
        return 0;
    return height(N->lchild) - height(N->rchild);
}

AVLTree* insertNode(AVLTree* node, int value) {
    if (node == NULL)
        return createNode(value);

    if (value < node->data)
        node->lchild = insertNode(node->lchild, value);
    else if (value > node->data)
        node->rchild = insertNode(node->rchild, value);
    else
        return node;

    node->height = 1 + max(height(node->lchild), height(node->rchild));

    int balance = getBalance(node);

    if (balance > 1 && value < node->lchild->data)
        return rightRotate(node);

    if (balance < -1 && value > node->rchild->data)
        return leftRotate(node);
```

```c
    if (balance > 1 && value > node->lchild->data) {
        node->lchild = leftRotate(node->lchild);
        return rightRotate(node);
    }

    if (balance < -1 && value < node->rchild->data) {
        node->rchild = rightRotate(node->rchild);
        return leftRotate(node);
    }

    return node;
}

void inorderTraversal(AVLTree* root) {
    if (root != NULL) {
        inorderTraversal(root->lchild);
        printf("%d ", root->data);
        inorderTraversal(root->rchild);
    }
}

void preorderTraversal(AVLTree* root) {
    if (root != NULL) {
        printf("%d ", root->data);
        preorderTraversal(root->lchild);
        preorderTraversal(root->rchild);
    }
}

void postorderTraversal(AVLTree* root) {
    if (root != NULL) {
        postorderTraversal(root->lchild);
        postorderTraversal(root->rchild);
        printf("%d ", root->data);
    }
}

int main() {
    int choice, value;

    while (1) {
        printf("\nMenu:\n");
        printf("1. Insert Node\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: ");
                scanf("%d", &value);
                root = insertNode(root, value);
                break;
            case 2:
                printf("Inorder Traversal: ");
                inorderTraversal(root);
                printf("\n");
                break;
            case 3:
                printf("Preorder Traversal: ");
                preorderTraversal(root);
                printf("\n");
                break;
            case 4:
                printf("Postorder Traversal: ");
                postorderTraversal(root);
                printf("\n");
                break;
            case 5:
                exit(0);
            default:
                printf("Invalid choice\n");
        }
    }
}
```

6. Fibonacci Search

```
#include <stdio.h>
int a[20], n, f[20], k, s;
int min(int x, int y)
{
    if (x<y)
        return x;
    else
        return y;
}
void read()
{
    printf("Enter the value n");
    scanf("%d", &n);
    printf("enter %d values", n);
    for(int i=0; i<n; i++)
```

```c
    {
        scanf("%d", &a[i]);
    }
    printf("enter target element");
    scanf("%d", &s);
}
int fibonacciseries()
{   k=1;
    f[0]=0;
    f[1]=1;
    while (f[k]<n)
    {
        k++;
        f[k] = f[k-1] + f[k-2];
    }
}
int fibonacciSearch()
{
    int offset=-1;
    while(1)
    {
        if (f[k-2]==0) break;
        int i = min(offset + f[k-2], n-1);
        if (s == a[i])
            return i;
        else if (s > a[i])
        {
            k = k-1;
            offset = i;
        }
        else k = k-2;
    }
    return -1;
}

int main(void)
{
    read();
    fibonacciseries();
    int ind = fibonacciSearch();
    if(ind>=0)
        printf("Found at index: %d",ind);
    else
        printf("%d isn't present in the array",s);
    return 0;
}
```

7. Quick Sort

```c
#include<stdio.h>

void swap(int *a, int *b){
    int temp=*a;
    *a=*b;
    *b=temp;
}

int partition(int arr[],int low, int high){
    int p = arr[low];
    int i = low;
    int j = high;

    while(i<j){
        while(arr[i]<=p&&i<=high-1){
            i++;
        }
        while(arr[j]>p&&j>=low+1){
            j--;
        }
        if(i<j){
            swap(&arr[i],&arr[j]);
        }
    }
    swap(&arr[low],&arr[j]);
    return j;
}

void quickSort(int arr[],int low,int high){
    if(low<high){
        int pivot = partition(arr,low,high);
        quickSort(arr,low,pivot-1);
        quickSort(arr,pivot+1,high);

    }
}

int main(){
    int arr[30],n;
    printf("Enter the number of elements: ");
    scanf("%d",&n);
    printf("Enter the elements: ");
    for(int i=0;i<n;i++){
        scanf("%d",&arr[i]);
    }
    quickSort(arr,0,n-1);
    printf("Sorted array: ");
    for(int i=0;i<n;i++){
        printf("%d ",arr[i]);
    }
}
```

8. Merge Sort

```
//merge sort
#include <stdio.h>
#include <stdlib.h>

void merge(int arr[],int left,int mid,int right){
    int i,j,k;
    int n1 = mid - left+1;
    int n2 = right - mid;

    int leftArr[n1],rightArr[n2];

    for(i=0;i<n1;i++){
        leftArr[i] = arr[left+i];
    }
    for(j=0;j<n2;j++){
        rightArr[j] = arr[mid+1+j];
    }
    i=0;
    j=0;
    k=left;
    while(i<n1&&j<n2){
        if(leftArr[i]<=rightArr[j]){
            arr[k]=leftArr[i];
            i++;
        }
        else{
            arr[k]=rightArr[j];
            j++;
        }
        k++;
    }
    while(i<n1){
        arr[k]=leftArr[i];
        i++;
        k++;
    }
    while(j<n2){
        arr[k]=rightArr[j];
        j++;
        k++;
    }
}

void mergeSort(int arr[],int left, int right){
    if(left<right){
        int mid = left + (right-left)/2;
        mergeSort(arr,left,mid);
        mergeSort(arr,mid+1,right);
        merge(arr,left,mid,right);
    }
}

int main(){
    int arr[30],n;
    printf("Enter no of elements: ");
    scanf("%d",&n);
    printf("Enter the elements: ");
    for(int i =0;i<n;i++){
        scanf("%d",&arr[i]);
    }
    mergeSort(arr,0,n-1);
    for(int i=0;i<n;i++){
        printf("%d ",arr[i]);
    }
}
```

9. Heap Sort

```
//heapsort
#include<stdio.h>
#include<stdlib.h>
void heapify(int arr[],int n,int i){
    int temp,maximum,left_index, right_index;
    maximum=i;
    right_index=2*i+2;
    left_index=2*i+1;
    if(left_index<n&&arr[left_index]>arr[maximum]){
        maximum=left_index;
    }
    if(right_index<n&&arr[right_index]>arr[maximum]){
        maximum=right_index;
    }
    if(maximum!=i){
        temp=arr[i];
        arr[i]=arr[maximum];
        arr[maximum]=temp;
        heapify(arr,n,maximum);
    }
}
void heapsort(int arr[],int n){
    int i,temp;
    for(i=n/2-1;i>=0;i--){
        heapify(arr,n,i);
    }
    for(i=n-1;i>0;i--){
        temp=arr[0];
        arr[0]=arr[i];
        arr[i]=temp;
        heapify(arr,i,0);
    }
}

int main(){
    int arr[30],n;
    printf("enter n ");
    scanf("%d",&n);
    printf("Enter the elements: ");
    for(int i=0;i<n;i++){
        scanf("%d",&arr[i]);
    }
    heapsort(arr,n);
    printf("Array after performing heap sort: ");
    for(int i=0;i<n;i++){
        printf("%d",arr[i]);
    }
}
```

10. Topological Sort

```
#include <stdio.h>

int main(){
int i,j,k,n,a[10][10],indeg[10],flag[10],count=0;

printf("Enter the no of vertices:\n");
scanf("%d",&n);

printf("Enter the adjacency matrix:\n");
for(i=0;i<n;i++){
printf("Enter row %d\n",i);
for(j=0;j<n;j++)
scanf("%d",&a[i][j]);
}
for(i=0;i<n;i++){
        indeg[i]=0;
        flag[i]=0;
    }

    for(i=0;i<n;i++)
        for(j=0;j<n;j++)
            indeg[i]=indeg[i]+a[j][i];

    printf("\nThe topological order is:");

    while(count<n)
    {
        for(k=0;k<n;k++)
        {
            if((indeg[k]==0) && (flag[k]==0))
            {
                printf("%d ",k);
                flag [k]=1;
                count++;
                for(i=0;i<n;i++)
                {
                   if(a[k][i]==1)
                      indeg[i]--;
                }
            }
         }


    }

    return 0;
}
```

11.BFS & DFS

```
#include<stdio.h>
int q[20],top=-1,front=-1,rear=-1,a[20][20],vis[20],stack[20];
int delete();
void add(int item);
void bfs(int s,int n);
void dfs(int s,int n);
void push(int item);
int pop();
void main()
{
int n,i,s,ch,j;
char c,dummy;
```

```c
printf("ENTER THE NUMBER VERTICES ");
scanf("%d",&n);
for(i=1;i<=n;i++)
{
      for(j=1;j<=n;j++)
      {
                printf("ENTER 1 IF %d HAS A NODE WITH %d ELSE 0 ",i,j);
               scanf("%d",&a[i][j]);
       }
}
printf("THE ADJACENCY MATRIX IS\n");
for(i=1;i<=n;i++)
{
      for(j=1;j<=n;j++)
     {
                 printf(" %d",a[i][j]);
      }
      printf("\n");
}
do
{
           for(i=1;i<=n;i++)
                 vis[i]=0;
          printf("\nMENU");
          printf("\n1.B.F.S");
          printf("\n2.D.F.S");
          printf("\nENTER YOUR CHOICE");
          scanf("%d",&ch);
          printf("ENTER THE SOURCE VERTEX :");
          scanf("%d",&s);

          switch(ch)
         {
                 case 1:bfs(s,n);
                              break;
                 case 2: dfs(s,n);
                              break;
          }
          printf("DO U WANT TO CONTINUE(Y/N) ? ");
          scanf("%c",&dummy);
          scanf("%c",&c);
}while((c=='y')||(c=='Y'));
}


//*************BFS(breadth-first search) code*************//
void bfs(int s,int n)
{
int p,i;
```

```
add(s);
vis[s]=1;
p=delete();
if(p!=0)
printf(" %d",p);
while(p!=0)
{
for(i=1;i<=n;i++)
if((a[p][i]!=0)&&(vis[i]==0))
{
     add(i);
     vis[i]=1;
}
p=delete();
if(p!=0)
printf(" %d",p);
}
}
void add(int item)
{
if(rear==19)
     printf("QUEUE FULL");
else
{
if(rear==-1)
{
    q[++rear]=item;
    front++;
}
else
    q[++rear]=item;
}
}
int delete()
{
int k;
if((front>rear)||(front==-1))
    return(0);
else
{
    k=q[front++];
    return(k);
}
}


//***************DFS(depth-first search) code******************//
void dfs(int s,int n)
{
```

```
int i,k;
push(s);
vis[s]=1;
k=pop();
if(k!=0)
    printf(" %d ",k);
while(k!=0)
{
    for(i=1;i<=n;i++)
    if((a[k][i]!=0)&&(vis[i]==0))
    {
        push(i);
        vis[i]=1;
    }
    k=pop();

    if(k!=0)
          printf(" %d ",k);
}
}
void push(int item)
{
    if(top==19)
        printf("Stack overflow ");
    else
        stack[++top]=item;
}
int pop()
{
    int k;
    if(top==-1)
        return(0);
    else
    {
        k=stack[top--];
        return(k);
    }
}
```