|  |
| --- |
| Assignment File of Data Structures And Algorithm |
|  |
|  |
|  |
|  |

Submitted to : Submitted by:

Dr. Rinkle Rani Chetan Singh Chauhan

MCA II( 3rd Sem)

201880014

**Assignment 5**

1. **Inoder, Preorder, Postorder Traversal**

#include <stdio.h>

#include <stdlib.h>

struct node {

int data;

struct node\* left;

struct node\* right; };

void inorder(struct node\* root){

if(root == NULL) return;

inorder(root->left);

printf("%d ->", root->data);

inorder(root->right); }

void preorder(struct node\* root){

if(root == NULL) return;

printf("%d ->", root->data);

preorder(root->left);

preorder(root->right); }

void postorder(struct node\* root) {

if(root == NULL) return;

postorder(root->left);

postorder(root->right);

printf("%d ->", root->data);}

struct node\* createNode(value){

struct node\* newNode = malloc(sizeof(struct node));

newNode->data = value;

newNode->left = NULL;

newNode->right = NULL;

return newNode; }

struct node\* insertLeft(struct node \*root, int value) {

root->left = createNode(value);

return root->left; }

struct node\* insertRight(struct node \*root, int value){

root->right = createNode(value);

return root->right; }

int main(){

struct node\* root = createNode(1);

insertLeft(root, 12);

insertRight(root, 9);

insertLeft(root->left, 5);

insertRight(root->left, 6);

printf("Inorder traversal \n");

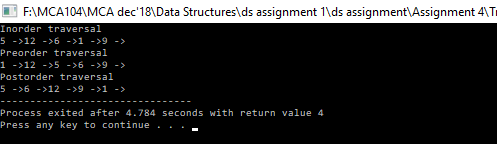
inorder(root);

printf("\nPreorder traversal \n");

preorder(root);

printf("\nPostorder traversal \n");

postorder(root); }



1. **Binary Search Tree**

#include<stdio.h>

#include<stdlib.h>

typedef struct BST

{

int data;

struct BST \*left;

struct BST \*right;

}node;

node \*create();

void insert(node \*,node \*);

void preorder(node \*);

int main(){

char ch;

node \*root=NULL,\*temp;

do{

temp=create();

if(root==NULL)

root=temp;

else

insert(root,temp);

printf("\nDo you want to enter more(y/n)?");

getchar();

scanf("%c",&ch);

}while(ch=='y'|ch=='Y');

printf("\nPreorder Traversal: ");

preorder(root);

return 0; }

node \*create() {

node \*temp;

printf("\nEnter data:");

temp=(node\*)malloc(sizeof(node));

scanf("%d",&temp->data);

temp->left=temp->right=NULL;

return temp; }

void insert(node \*root,node \*temp) {

if(temp->data<root->data) {

if(root->left!=NULL)

insert(root->left,temp);

else

root->left=temp; }

if(temp->data>root->data) {

if(root->right!=NULL)

insert(root->right,temp);

else

root->right=temp; } }

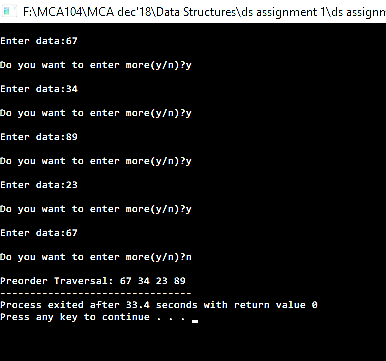
void preorder(node \*root) {

if(root!=NULL) {

printf("%d ",root->data);

preorder(root->left);

preorder(root->right); } }



1. **Binary Search Tree deletion**

#include<stdio.h>

#include<stdlib.h>

typedef struct node{

int data;

struct node \*pre, \*nex;

}node;

node \*allocateMemory(int d);

node \*createBst(node \*head, int a);

void inOrder(node \*head);

node \*minNode(node \*temp);

node \*deleteNode(node \*head, int d);

int main(){

node \*head = NULL;

int a;

head = createBst(head, 5);

createBst(head, 3);

createBst(head, 7);

createBst(head, 2);

createBst(head, 4);

createBst(head, 6);

createBst(head, 8);

inOrder(head);

printf("\nDeleting node with data 3\n");

head = deleteNode(head, 3);

printf("Inorder traversal of the modified tree \n");

inOrder(head);

printf("\nDeleting node with data 7\n");

head = deleteNode(head, 7);

printf("Inorder traversal of the modified tree \n");

inOrder(head);

printf("\nDeleting node with data 5\n");

head = deleteNode(head, 5);

printf("Inorder traversal of the modified tree \n");

inOrder(head);

return 0; }

node \*deleteNode(node \*head, int d){

if(head == NULL)

return head;

if(d < head -> data)

head -> pre = deleteNode(head -> pre, d);

else if(d > head -> data)

head -> nex = deleteNode(head -> nex, d);

else{

if(head -> pre == NULL){

node \*temp = head -> nex;

free(head);

return temp; }

else if(head -> nex == NULL){

node \*temp = head -> pre;

free(head);

return temp; }

node \*temp = minNode(head -> nex);

head -> data = temp -> data;

head -> nex = deleteNode(head -> nex, temp -> data); }

return head; }

node \*minNode(node \*temp){

while (temp -> pre != NULL)

temp = temp -> pre;

return temp; }

node \*createBst(node \*head, int a){

if(head == NULL)

return allocateMemory(a);

if(a < head -> data)

head -> pre = createBst(head -> pre, a);

else if(a > head -> data)

head -> nex = createBst(head -> nex, a);

return head; }

node \*allocateMemory(int d){

node \*temp = (node\*)malloc(sizeof(node));

temp -> data = d;

temp -> pre = NULL;

temp -> nex = NULL;

return temp; }

void inOrder(node \*head){

if(head == NULL)

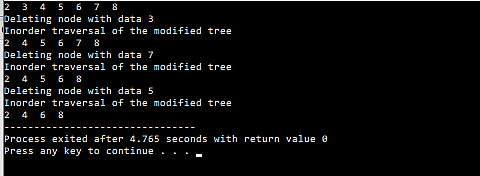
return;

else{

inOrder(head -> pre);

printf("%d ", head -> data);

inOrder(head -> nex); } }



**Assignment 6**

**Dijkastra Algorithm**

#include<stdio.h>

#include<conio.h>

#define INFINITY 9999

#define MAX 10

void dijkstra(int G[MAX][MAX],int n,int startnode);

int main() {

int G[MAX][MAX],i,j,n,u;

printf("Enter no. of vertices:");

scanf("%d",&n);

printf("\nEnter the adjacency matrix:\n");

for(i=0;i<n;i++)

for(j=0;j<n;j++)

scanf("%d",&G[i][j]);

printf("\nEnter the starting node:");

scanf("%d",&u);

dijkstra(G,n,u);

return 0; }

void dijkstra(int G[MAX][MAX],int n,int startnode)

{ int cost[MAX][MAX],distance[MAX],pred[MAX];

int visited[MAX],count,mindistance,nextnode,i,j;

for(i=0;i<n;i++)

for(j=0;j<n;j++)

if(G[i][j]==0)

cost[i][j]=INFINITY;

else

cost[i][j]=G[i][j];

for(i=0;i<n;i++) {

distance[i]=cost[startnode][i];

pred[i]=startnode;

visited[i]=0; }

distance[startnode]=0;

visited[startnode]=1;

count=1;

while(count<n-1) {

mindistance=INFINITY;

for(i=0;i<n;i++)

if(distance[i]<mindistance&&!visited[i]) {

mindistance=distance[i];

nextnode=i; }

visited[nextnode]=1;

for(i=0;i<n;i++)

if(!visited[i])

if(mindistance+cost[nextnode][i]<distance[i])

{

distance[i]=mindistance+cost[nextnode][i];

pred[i]=nextnode; }

count++; }

for(i=0;i<n;i++)

if(i!=startnode) {

printf("\nDistance of node%d=%d",i,distance[i]);

printf("\nPath=%d",i);

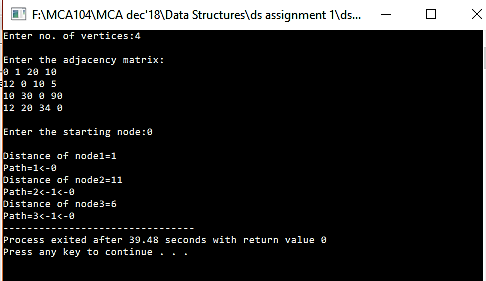
j=i;

do{

j=pred[j];

printf("<-%d",j);

}while(j!=startnode); }}



**2. Floyd Warshall Algorithm**

#include<stdio.h>

#include<conio.h>

#include<math.h>

int max(int,int);

void warshal(int p[10][10],int n) {

int i,j,k;

for(k=1;k<=n;k++)

for(i=1;i<=n;i++)

for(j=1;j<=n;j++)

p[i][j]=max(p[i][j],p[i][k]&&p[k][j]); }

int max(int a,int b) { ;

if(a>b)

return(a);

else

return(b); }

void main() {

int p[10][10]={0},n,e,u,v,i,j;

clrscr();

printf("n Enter the number of vertices:");

scanf("%d",&n);

printf("n Enter the number of edges:");

scanf("%d",&e);

for(i=1;i<=e;i++) {

printf("n Enter the end vertices of edge %d:",i);

scanf("%d%d",&u,&v);

p[u][v]=1; }

printf("n Matrix of input data: n");

for(i=1;i<=n;i++) {

for(j=1;j<=n;j++)

printf("%dt",p[i][j]);

printf("n"); }

warshal(p,n);

printf("n Transitive closure: n");

for(i=1;i<=n;i++) {

for(j=1;j<=n;j++)

printf("%dt",p[i][j]);

printf("n"); }

getch(); }

**3.Breadth First Search**

#include<stdio.h>

#include<stdlib.h>

#define MAX 100

#define initial 1

#define waiting 2

#define visited 3

int n;

int adj[MAX][MAX];

int state[MAX];

void create\_graph();

void BF\_Traversal();

void BFS(int v);

int queue[MAX], front = -1,rear = -1;

void insert\_queue(int vertex);

int delete\_queue();

int isEmpty\_queue();

int main() {

create\_graph();

BF\_Traversal();

return 0; }

void BF\_Traversal() {

int v;

for(v=0; v<n; v++)

state[v] = initial;

printf("Enter Start Vertex for BFS: \n");

scanf("%d", &v);

BFS(v); }

void BFS(int v) {

int i;

insert\_queue(v);

state[v] = waiting;

while(!isEmpty\_queue()) {

v = delete\_queue( );

printf("%d ",v);

state[v] = visited;

for(i=0; i<n; i++) {

if(adj[v][i] == 1 && state[i] == initial) {

insert\_queue(i);

state[i] = waiting; } } }

printf("\n"); }

void insert\_queue(int vertex) {

if(rear == MAX-1)

printf("Queue Overflow\n");

else {

if(front == -1)

front = 0;

rear = rear+1;

queue[rear] = vertex ; } }

int isEmpty\_queue(){

if(front == -1 || front > rear)

return 1;

else

return 0; }

int delete\_queue(){

int delete\_item;

if(front == -1 || front > rear) {

printf("Queue Underflow\n");

exit(1); }

delete\_item = queue[front];

front = front+1;

return delete\_item; }

void create\_graph() {

int count,max\_edge,origin,destin;

printf("Enter number of vertices : ");

scanf("%d",&n);

max\_edge = n\*(n-1);

for(count=1; count<=max\_edge; count++) {

printf("Enter edge %d( -1 -1 to quit ) : ",count);

scanf("%d %d",&origin,&destin);

if((origin == -1) && (destin == -1))

break;

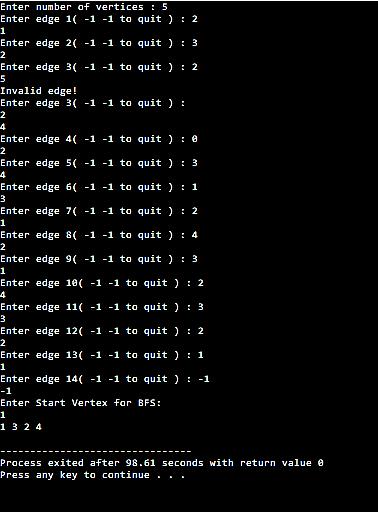
if(origin>=n || destin>=n || origin<0 || destin<0) {

printf("Invalid edge!\n");

count--; }

else{

adj[origin][destin] = 1; } } }



1. **Kruskal’s Algorithm**

#include<stdio.h>

#define MAX 30

typedef struct edge {

int u,v,w;

}edge;

typedef struct edgelist {

edge data[MAX];

int n;

}edgelist;

edgelist elist;

int G[MAX][MAX],n;

edgelist spanlist;

void kruskal();

int find(int belongs[],int vertexno);

void union1(int belongs[],int c1,int c2);

void sort();

void print();

void main(){

int i,j,total\_cost;

printf("\nEnter number of vertices:");

scanf("%d",&n);

printf("\nEnter the adjacency matrix:\n");

for(i=0;i<n;i++)

for(j=0;j<n;j++)

scanf("%d",&G[i][j]);

kruskal();

print(); }

void kruskal() {

int belongs[MAX],i,j,cno1,cno2;

elist.n=0;

for(i=1;i<n;i++)

for(j=0;j<i;j++){

if(G[i][j]!=0) {

elist.data[elist.n].u=i;

elist.data[elist.n].v=j;

elist.data[elist.n].w=G[i][j];

elist.n++;}}

sort();

for(i=0;i<n;i++)

belongs[i]=i;

spanlist.n=0;

for(i=0;i<elist.n;i++){

cno1=find(belongs,elist.data[i].u);

cno2=find(belongs,elist.data[i].v);

if(cno1!=cno2) {

spanlist.data[spanlist.n]=elist.data[i];

spanlist.n=spanlist.n+1;

union1(belongs,cno1,cno2); } }}

int find(int belongs[],int vertexno){

return(belongs[vertexno]); }

void union1(int belongs[],int c1,int c2); {

int i;

for(i=0;i<n;i++)

if(belongs[i]==c2)

belongs[i]=c1; }

void sort(){

int i,j;

edge temp;

for(i=1;i<elist.n;i++)

for(j=0;j<elist.n-1;j++)

if(elist.data[j].w>elist.data[j+1].w) {

temp=elist.data[j];

elist.data[j]=elist.data[j+1];

elist.data[j+1]=temp; } }

void print() {

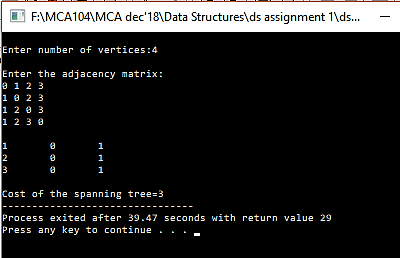
int i,cost=0;

for(i=0;i<spanlist.n;i++) {

printf("\n%d\t%d\t%d",spanlist.data[i].u,spanlist.data[i].v,spanlist.data[i].w);

cost=cost+spanlist.data[i].w; }

printf("\n\nCost of the spanning tree=%d",cost); }



**Assignment 7**

**Hashing**

#include <stdio.h>

#include <stdlib.h>

#define SIZE 20

struct DataItem {

int data;

int key;

};

struct DataItem\* hashArray[SIZE];

struct DataItem\* dummyItem;

struct DataItem\* item;

int hashCode(int key) {

return key % SIZE; }

struct DataItem \*search(int key) {

int hashIndex = hashCode(key);

while(hashArray[hashIndex] != NULL) {

if(hashArray[hashIndex]->key == key)

return hashArray[hashIndex];

++hashIndex;

hashIndex %= SIZE; }

return NULL; }

void insert(int key,int data) {

struct DataItem \*item = (struct DataItem\*) malloc(sizeof(struct DataItem));

item->data = data;

item->key = key;

int hashIndex = hashCode(key);

while(hashArray[hashIndex] != NULL && hashArray[hashIndex]->key != -1) {

++hashIndex;

hashIndex %= SIZE; }

hashArray[hashIndex] = item; }

struct DataItem\* delete(struct DataItem\* item) {

int key = item->key;

int hashIndex = hashCode(key);

while(hashArray[hashIndex] != NULL) {

if(hashArray[hashIndex]->key == key) {

struct DataItem\* temp = hashArray[hashIndex];

hashArray[hashIndex] = dummyItem;

return temp; }

++hashIndex;

hashIndex %= SIZE; }

return NULL; }

void display() {

int i = 0;

for(i = 0; i<SIZE; i++) {

if(hashArray[i] != NULL)

printf(" (%d,%d)",hashArray[i]->key,hashArray[i]->data);

else

printf(" ~~ ");

}

printf("\n"); }

int main() {

dummyItem = (struct DataItem\*) malloc(sizeof(struct DataItem));

dummyItem->data = -1;

dummyItem->key = -1;

insert(1, 20);

insert(2, 70);

insert(42, 80);

insert(4, 25);

insert(12, 44);

insert(14, 32);

insert(17, 11);

insert(13, 78);

insert(37, 97);

display();

item = search(37);

if(item != NULL) {

printf("Element found: %d\n", item->data);

} else {

printf("Element not found\n");}

delete(item);

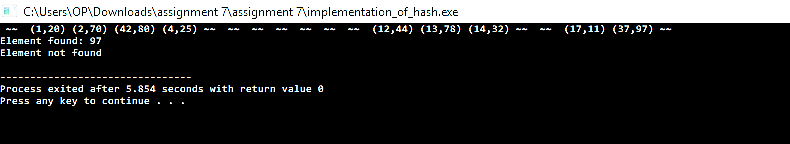
item = search(37);

if(item != NULL) {

printf("Element found: %d\n", item->data);

} else {

printf("Element not found\n");} }



**Shell Sort**

#include <stdio.h>

void shellsort(int arr[], int num){

int i, j, k, tmp;

for (i = num / 2; i > 0; i = i / 2){

for (j = i; j < num; j++){

for(k = j - i; k >= 0; k = k - i){

if (arr[k+i] >= arr[k])

break;

else{

tmp = arr[k];

arr[k] = arr[k+i];

arr[k+i] = tmp;} } } } }

int main() {

int arr[30];

int k, num;

printf("Enter total no. of elements : ");

scanf("%d", &num);

printf("\nEnter %d numbers: ", num);

for (k = 0 ; k < num; k++){

scanf("%d", &arr[k]); }

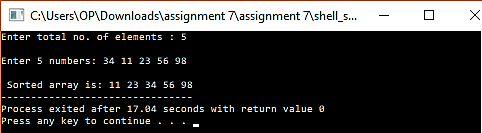
shellsort(arr, num);

printf("\n Sorted array is: ");

for (k = 0; k < num; k++)

printf("%d ", arr[k]);

return 0; }



**Radix Sort**

#include<stdio.h>

int largest(int a[], int n) {

int large = a[0], i;

for(i = 1; i < n; i++) {

if(large < a[i])

large = a[i]; }

return large; }

void RadixSort(int a[], int n) {

int bucket[10][10], bucket\_count[10];

int i, j, k, remainder, NOP=0, divisor=1, large, pass;

large = largest(a, n);

printf("The large element %d\n",large);

while(large > 0){

NOP++;

large/=10; }

for(pass = 0; pass < NOP; pass++){

for(i = 0; i < 10; i++) {

bucket\_count[i] = 0; }

for(i = 0; i < n; i++) {

remainder = (a[i] / divisor) % 10;

bucket[remainder][bucket\_count[remainder]] = a[i];

bucket\_count[remainder] += 1; }

i = 0;

for(k = 0; k < 10; k++) {

for(j = 0; j < bucket\_count[k]; j++) {

a[i] = bucket[k][j];

i++; } }

divisor \*= 10;

for(i = 0; i < n; i++)

printf("%d ",a[i]);

printf("\n"); } }

int main() {

int i, n, a[10];

printf("Enter the number of elements :: ");

scanf("%d",&n);

printf("Enter the elements :: ");

for(i = 0; i < n; i++) {

scanf("%d",&a[i]); }

RadixSort(a,n);

printf("The sorted elements are :: ");

for(i = 0; i < n; i++)

printf("%d ",a[i]);

printf("\n");

return 0; }

