**Analysis and Design of Algorithms**

**1.Write program to do the following:**

**a. Print all the nodes reachable from a given starting node in a digraph using**

**BFS method.**

**b. Check whether a given graph is connected or not using DFS method**.

#include<stdio.h>

#include<conio.h>

int Q[10],f=0,r=0;

int vis[10],a[10][10];

void bfs(int v,int n){

vis[v]=1;

Q[r]=v;

while(f<=r){

int u=Q[f];

printf("%d",u);

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

if(a[u][i]==1 && vis[i]==0){

r=r+1;

Q[r]=i;

vis[i]=1;

}

}

f=f+1;

}

}

void main()

{

int n,begin;

int m,c,d;

printf("Enter the number of vertices");

scanf("%d",&n);

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

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

a[i][j]=0;

}

}

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

scanf("%d",&m);

for(int i=1;i<=m;i++){

printf("Enter the edges");

scanf("%d%d",&c,&d);

a[c][d]=1;

}

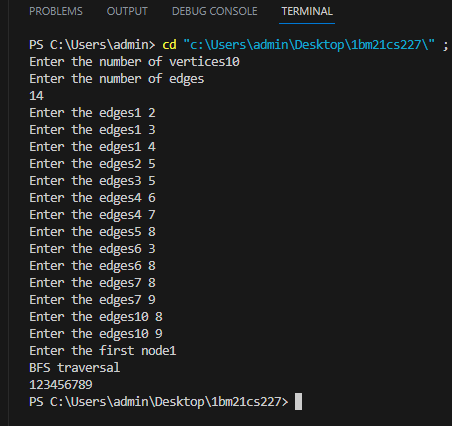
printf("Enter the first node");

scanf("%d",&begin);

printf("BFS traversal\n");

bfs(begin,n);

}



**2. Write program to obtain the Topological ordering of vertices in a given**

**digraph.**

#include<stdio.h>

void dfs(int);

int a[10][10],n,exp[10],vis[10],j=0;

void main(){

int m,u,v;

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

scanf("%d",&n);

printf("Enter number of edges edges\n");

scanf("%d",&m);

printf("Enter all edges\n");

for(int i=1;i<=m;i++){

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

a[u][v]=1;

}

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

if(vis[i]==0){

dfs(i);

}

}

printf("Topological order\n");

for(int i=n-1;i>=0;i--){

printf("%d\t",exp[i]);

}

}

void dfs(int v){

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

if(a[v][i]==1 && vis[i]==0){

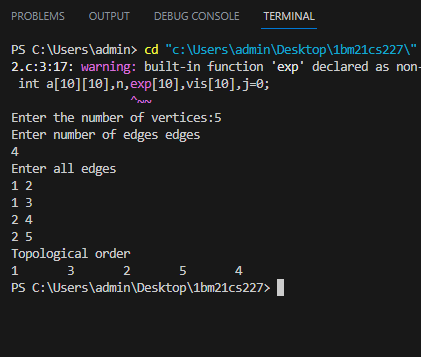
dfs(i);

}

}

exp[j++]=v;

}



**3. Implement Johnson Trotter algorithm to generate permutations.**

#include <stdio.h>

#include <conio.h>

int LEFT\_TO\_RIGHT = 1;

int RIGHT\_TO\_LEFT = 0;

// position of mobile element

int searchArr(int a[], int n, int mobile)

{

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

if (a[i] == mobile)

return i + 1;

}

// getting mobile element

int getMobile(int a[], int dir[], int n)

{

int mobile\_prev = 0, mobile = 0;

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

{

if (dir[a[i] - 1] == RIGHT\_TO\_LEFT && i != 0)

{

if (a[i] > a[i - 1] && a[i] > mobile\_prev)

{

mobile = a[i];

mobile\_prev = mobile;

}

}

if (dir[a[i] - 1] == LEFT\_TO\_RIGHT && i != n - 1)

{

if (a[i] > a[i + 1] && a[i] > mobile\_prev)

{

mobile = a[i];

mobile\_prev = mobile;

}

}

}

// if mobile element or large element found at 0 or n-1 return 0

if (mobile == 0 && mobile\_prev ==0)

return 0;

else

return mobile;

}

// printing one permutation

int printOnePerm(int a[], int dir[], int n)

{

int mobile = getMobile(a, dir, n);

int pos = searchArr(a, n, mobile);

if (dir[a[pos - 1] - 1] == RIGHT\_TO\_LEFT)

{

printf("\n");

// swap

int temp;

temp = a[pos - 1];

a[pos - 1] = a[pos - 2];

a[pos - 2] = temp;

}

else if (dir[a[pos - 1] - 1] == LEFT\_TO\_RIGHT)

{

printf("\n");

int temp;

temp = a[pos];

a[pos] = a[pos - 1];

a[pos - 1] = temp;

}

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

{

if (a[i] > mobile)

{

if (dir[a[i] - 1] ==

LEFT\_TO\_RIGHT)

dir[a[i] - 1] =

RIGHT\_TO\_LEFT;

else if (dir[a[i] - 1] ==

RIGHT\_TO\_LEFT)

dir[a[i] - 1] =

LEFT\_TO\_RIGHT;

}

}

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

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

}

// factorial

int fact(int n)

{

int res = 1;

int i;

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

res = res \* i;

return res;

}

void printPermutation(int n)

{

int a[n];

int dir[n];

printf("\n");

printf("\n");

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

{

a[i] = i + 1;

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

printf("\n");

}

printf("\n");

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

dir[i] = RIGHT\_TO\_LEFT;

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

printOnePerm(a, dir, n);

printf("\n");

}

int main()

{

int n;

printf("\n Please Enter the value of n");

scanf("%d", &n);

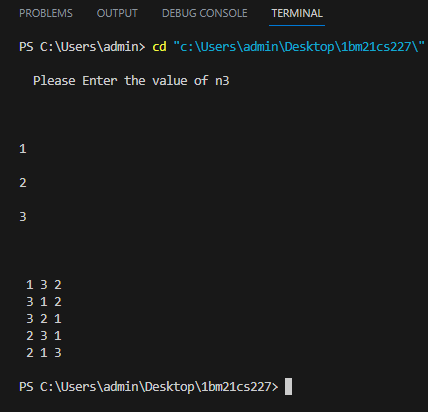
printf("\n");

printPermutation(n);

printf("\n");

return 0;

}

****

**4.** **Sort a given set of N integer elements using Merge Sort technique and**

**compute its time taken. Run the program for different values of N and recordthe time taken to sort.**

#include <stdio.h>

#include <stdlib.h>

void merge(int A[], int si, int mid, int ei)

{

int i, j, k, c[100];

i = si;

j = mid + 1;

k = si;

while (i <= mid && j <= ei)

{

if (A[i] < A[j])

c[k++] = A[i++];

else

c[k++] = A[j++];

}

while (i <= mid)

c[k++] = A[i++];

while (j <= ei)

c[k++] = A[j++];

for (i = si; i <= ei; i++)

{

A[i] = c[i];

}

}

void mergeSort(int A[], int si, int ei)

{

int mid;

if (si < ei)

{

mid = (si + ei) / 2;

mergeSort(A, si, mid);

mergeSort(A, mid + 1, ei);

merge(A, si, mid, ei);

}

}

int main()

{

int n;

int A[10];

printf("\nEnter the number of elements: ");

scanf("%d", &n);

printf("Enter array elements\n");

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

{

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

}

printf("Array Elements: \n");

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

{

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

}

printf("\nSorted Array: \n");

mergeSort(A, 0, n - 1);

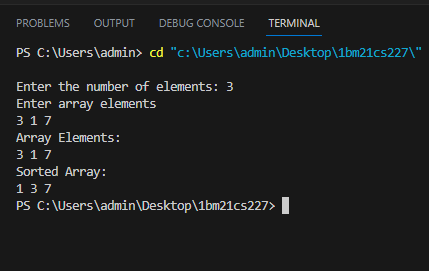
for (int i = 0; i < n; i++)

{

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

}

}



**5.** **Sort a given set of N integer elements using Quick Sort technique and**

**compute its time taken.**

//Quick\_sort

// C code to implement quicksort

#include <stdio.h>

# include<time.h>

void swap(int\* a, int\* b)

{

int t = \*a;

\*a = \*b;

\*b = t;

}

int partition(int arr[], int low, int high)

{

int pivot = arr[high];

int i = (low - 1);

for (int j = low; j <= high - 1; j++) {

if (arr[j] < pivot) {

i++;

swap(&arr[i], &arr[j]);

}

}

swap(&arr[i + 1], &arr[high]);

return (i + 1);

}

void quickSort(int arr[], int low, int high)

{

if (low < high) {

int pi = partition(arr, low, high);

quickSort(arr, low, pi - 1);

quickSort(arr, pi + 1, high);

}

}

int main()

{

clock\_t start, end;

double execution\_time;

start = clock();

int arr[] = { 10, 7, 8, 9, 1, 5 };

int N = sizeof(arr) / sizeof(arr[0]);

quickSort(arr, 0, N - 1);

printf("Sorted array: \n");

for (int i = 0; i < N; i++)

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

end = clock();

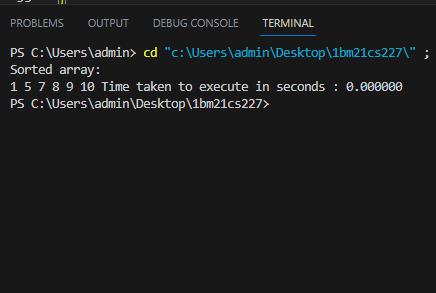
/\* Get the time taken by program to execute in seconds \*/

double duration = ((double)end - start)/CLOCKS\_PER\_SEC;

printf("Time taken to execute in seconds : %f", duration);

return 0;

}



**6.** **Sort a given set of N integer elements using Heap Sort technique and**

**compute its time taken.**

#include <stdio.h>

void heapify(int arr[], int i, int size)

{

int left = 2 \* i + 1;

int right = 2 \* i + 2;

int maxIdx = i;

if (left < size && arr[left] > arr[maxIdx])

{

maxIdx = left;

}

if (right < size && arr[right] > arr[maxIdx])

{

maxIdx = right;

}

if (maxIdx != i)

{

int temp = arr[maxIdx];

arr[maxIdx] = arr[i];

arr[i] = temp;

heapify(arr, maxIdx, size);

}

}

void heapSort(int arr[], int n)

{

for (int i = n / 2; i >= 0; i--)

{

heapify(arr, i, n);

}

for (int i = n - 1; i >= 0; i--)

{

int temp = arr[0];

arr[0] = arr[i];

arr[i] = temp;

heapify(arr, 0, i);

}

}

int main()

{

int n;

printf("\nEnter the number of elements: ");

scanf("%d", &n);

int arr[n];

printf("Enter array elements: ");

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

{

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

}

heapSort(arr, n);

printf("Sorted Array:\n");

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

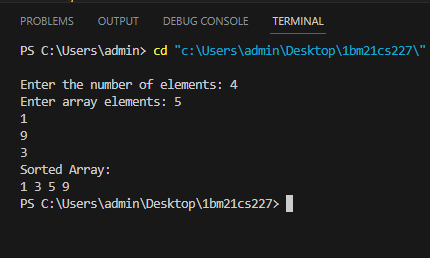
{

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

}

return 0;

}



**7.** **Implement 0/1 Knapsack problem using dynamic programming.**

#include <stdio.h>

#include <string.h>

int findMax(int n1, int n2){

if(n1>n2) {

return n1;

} else {

return n2;

}

}

int knapsack(int W, int wt[], int val[], int n){

int K[n+1][W+1];

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

for(int w = 0; w<=W; w++) {

// if any one of object not selected or weight is zero then no profit

if(i == 0 || w == 0) {

K[i][w] = 0;

}

else if (wt[i-1] <= w) {

K[i][w] = findMax(val[i-1] + K[i-1][w-wt[i-1]], K[i-1][w]);

} else {

K[i][w] = K[i-1][w];

}

}

}

return K[n][W];

}

int main(){

// profit

int val[5] = {12,10,20,15};

// weight

int wt[5] = {2,1,3,2};

// max capacity

int W = 5;

// length of array

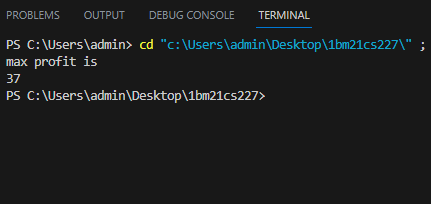
int len = sizeof val / sizeof val[0];

printf("max profit is\n");

int myprofit=knapsack(W,wt,val,len);

printf("%d",myprofit);

}



**8.** **Implement All Pair Shortest paths problem using Floyd’s algorithm.**

#include<stdio.h>

void floyd(int a[4][4], int n)

{

for(int k=0;k<n;k++)

{

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

{

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

{

if(a[i][j]>a[i][k]+a[k][j])

{

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

}

}

}

}

printf("All Pairs Shortest Path is :\n");

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

{

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

{

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

}

printf("\n");

}

}

int main()

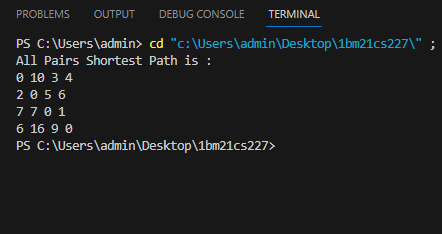
{

int cost[4][4] = {{0,999,3,999}, {2,0,999,999}, {999,7,0,1}, {6,999,999,0}};

int n = 4;

floyd(cost,n);

}



**9.** **Find Minimum Cost Spanning Tree of a given undirected graph using**

**Prim/Kruskal’s algorithm.**

#include <stdio.h>

#include <stdbool.h>

#define MAX\_VERTICES 10

int n;

int cost[MAX\_VERTICES][MAX\_VERTICES];

int f[MAX\_VERTICES][2];

int sum = 0;

void prim() {

int selected[MAX\_VERTICES] = {false};

int numSelected = 0;

int x, y;

selected[0] = true;

numSelected++;

while (numSelected < n) {

int min = 999;

int u, v;

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

if (selected[i]) {

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

if (!selected[j] && cost[i][j] != 0 && cost[i][j] < min) {

min = cost[i][j];

u = i;

v = j;

}

}

}

}

selected[v] = true;

f[numSelected - 1][0] = u;

f[numSelected - 1][1] = v;

numSelected++;

sum += cost[u][v];

}

}

int main() {

printf("Enter the number of vertices (n, max 10): ");

scanf("%d", &n);

printf("Enter the edges (adjacency matrix):\n");

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

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

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

}

}

prim();

printf("\nPrim's algorithm:\n");

printf("Minimum spanning tree:\n");

for (int i = 0; i < n - 1; i++) {

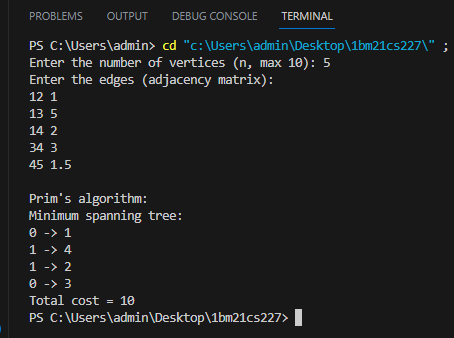
printf("%d -> %d\n", f[i][0], f[i][1]);

}

printf("Total cost = %d\n", sum);

return 0;

}



**10.Kruskal**

#include <stdio.h>

#include <stdbool.h>

#define MAX\_VERTICES 10

int n, parent[MAX\_VERTICES];

int f[MAX\_VERTICES][2];

int cost[MAX\_VERTICES][MAX\_VERTICES];

int sum = 0;

void unionn(int a, int b) {

int i = find(a);

int j = find(b);

if (i != j) {

parent[j] = i;

}

}

int find(int a) {

while (parent[a] != a) {

a = parent[a];

}

return a;

}

void kruskal() {

int count = 0;

int k = 0;

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

parent[i] = i;

}

while (count != n - 1) {

int min = 999;

int u, v;

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

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

if (cost[i][j] < min && cost[i][j] != 0) {

min = cost[i][j];

u = i;

v = j;

}

}

}

int x = find(u);

int y = find(v);

if (x != y) {

f[k][0] = u;

f[k][1] = v;

k++;

count++;

sum += cost[u][v];

unionn(x, y);

}

cost[u][v] = cost[v][u] = 999;

}

}

int main() {

printf("Enter the number of vertices (n, max 10): ");

scanf("%d", &n);

printf("Enter the edges (adjacency matrix):\n");

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

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

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

}

}

kruskal();

printf("\nKruskal's algorithm:\n");

printf("Minimum spanning tree:\n");

for (int i = 0; i < n - 1; i++) {

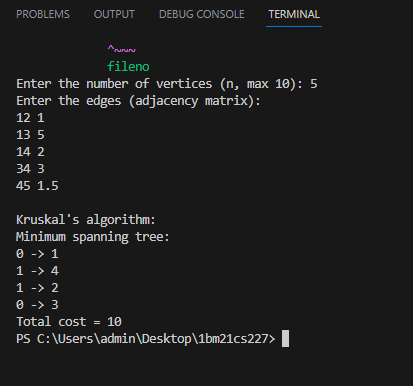
printf("%d -> %d\n", f[i][0], f[i][1]);

}

printf("Total cost = %d\n", sum);

return 0;

}

****

**11.** **From a given vertex in a weighted connected graph, find shortest paths to other vertices using Dijkstra’s algorithm.**

#include <stdio.h>

#include <stdbool.h>

#define MAX\_VERTICES 10

#define INF 9999

int n;

int graph[MAX\_VERTICES][MAX\_VERTICES];

int distance[MAX\_VERTICES];

bool visited[MAX\_VERTICES];

int findMinDistance() {

int min = INF;

int minIndex;

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

if (!visited[v] && distance[v] < min) {

min = distance[v];

minIndex = v;

}

}

return minIndex;

}

void dijkstra(int startVertex) {

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

distance[i] = INF;

visited[i] = false;

}

distance[startVertex] = 0;

for (int count = 0; count < n - 1; count++) {

int u = findMinDistance();

visited[u] = true;

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

if (!visited[v] && graph[u][v] && distance[u] + graph[u][v] < distance[v]) {

distance[v] = distance[u] + graph[u][v];

}

}

}

}

int main() {

printf("Enter the number of vertices (n, max 10): ");

scanf("%d", &n);

printf("Enter the weighted adjacency matrix (enter 0 for no edge):\n");

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

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

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

}

}

int startVertex;

printf("Enter the starting vertex (0 to %d): ", n - 1);

scanf("%d", &startVertex);

dijkstra(startVertex);

printf("\nShortest paths from vertex %d to all other vertices:\n", startVertex);

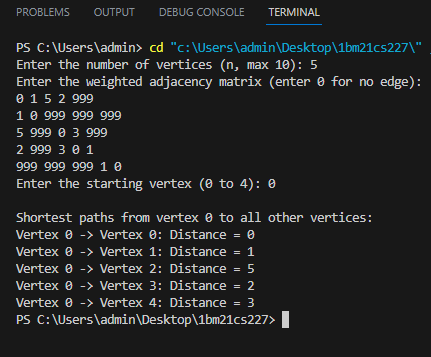
for (int i = 0; i < n; i++) {

printf("Vertex %d -> Vertex %d: Distance = %d\n", startVertex, i, distance[i]);

}

return 0;

}



**12.** **Implement “N-Queens Problem” using Backtracking.**

#include <stdio.h>

int n, count=0;

int isSafe(char board[n][n], int row, int col)

{

for (int i = row - 1; i >= 0; i--)

{

if (board[i][col] == 'Q')

{

return 0;

}

}

for (int i = row - 1, j = col - 1; i >= 0 && j >= 0; i--, j--)

{

if (board[i][j] == 'Q')

{

return 0;

}

}

for (int i = row - 1, j = col + 1; i >= 0 && j < n; i--, j++)

{

if (board[i][j] == 'Q')

{

return 0;

}

}

return 1;

}

void printBoard(char board[][n])

{

printf("\n---Chess Board---\n");

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

{

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

{

printf("%c ", board[i][j]);

}

printf("\n");

}

}

void nQueens(char board[n][n], int row)

{

if (row == n)

{

printBoard(board);

count++;

return;

}

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

{

if (isSafe(board, row, j) == 1)

{

board[row][j] = 'Q';

nQueens(board, row + 1);

board[row][j] = 'X';

}

}

}

int main()

{

printf("Enter the size of the board: ");

scanf("%d", &n);

char board[n][n];

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

{

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

{

board[i][j] = 'X';

}

}

nQueens(board, 0);

printf("\nTotal Possible Solution: %d ",count);

}

