```cpp
#include <bits/stdc++.h>
using namespace std;

// Function to create an adjacency list representation of the graph
void prepareadjlist(unordered_map<int, list<int>>& adjlist, vector<pair<int,
int>>& edges) {
    for (int i = 0; i < edges.size(); i++) {
        int u = edges[i].first; // Extracting the first node of the edge
        int v = edges[i].second; // Extracting the second node of the edge

        // Adding edges to the adjacency list (undirected graph)
        adjlist[u].push_back(v);
        adjlist[v].push_back(u);
    }
}

// Breadth-First Search (BFS) algorithm
void bfs(unordered_map<int, list<int>>& adjlist, unordered_map<int, bool>&
visited, vector<int>& ans, int node) {
    queue<int> q; // Queue for BFS traversal
    q.push(node); // Push the starting node into the queue

    visited[node] = true; // Mark the starting node as visited

    while (!q.empty()) { // While there are nodes in the queue
        int frontnode = q.front(); // Get the front node from the queue
        q.pop(); // Remove the front node from the queue

        ans.push_back(frontnode); // Add the front node to the result vector

        // Traverse all adjacent nodes of the current node
        for (auto i : adjlist[frontnode]) {
            if (!visited[i]) {
                q.push(i); // Push unvisited adjacent nodes into the queue
                visited[i] = true; // Mark them as visited
            }
        }
    }
}

// Function to perform BFS traversal on the graph
vector<int> BFS(int vertex, vector<pair<int, int>> edge) {
    unordered_map<int, list<int>> adjlist; // Adjacency list representation of
the graph
    vector<int> ans; // Vector to store the BFS traversal result
    unordered_map<int, bool> visited; // Map to keep track of visited nodes
```

```
    prepareadjlist(adjlist, edge); // Create the adjacency list representation

    // Iterate through all nodes and perform BFS if not visited before
    for (int i = 0; i < vertex; i++) {
        if (!visited[i]) {
            bfs(adjlist, visited, ans, i); // Call BFS for unvisited nodes
        }
    }

    return ans; // Return the BFS traversal result
}

// Main function
int main() {
    int V = 5; // Number of vertices
    vector<pair<int, int>> edges = {{0, 1}, {0, 2}, {1, 3}, {2, 4}}; // Edges
of the graph

    vector<int> result = BFS(V, edges); // Perform BFS traversal

    // Output the BFS traversal result
    cout << "BFS Traversal: ";
    for (int i = 0; i < result.size(); i++) {
        cout << result[i] << " ";
    }
    cout << endl;

    return 0;
}
```

```cpp
#include <bits/stdc++.h>
using namespace std;
void createadj(  unordered_map<int,list<int>>
&adj,vector<pair<int,int>>&edges){


    for(int i=0;i<edges.size();i++){
       int u=edges[i][0];
        int v=edges[i][1];
        adj[u].push_back(v);
        adj[v].push_back(u);

    }
}
void
bfs(unordered_map<int,list<int>>&adj,unordered_map<int,bool>&visited,vector<in
t>&ans,int i){

    queue<int>q;
    q.push(i);
    visited[i]=true;
    while(!q.empty()){
        int frontnode=i;
        q.pop();
        ans.push_back(frontnode);
        for(j:adj[frontnode]){
            if(![visited[j]]){
                visited[j]=true;
               q.push(i);
            }
        }

    }


}



void BFS(int vertex,vector<pair<int,int>>&edges)
{
    unordered_map<int,list<int>> adj;
    vector<int> ans;
    unordered_map<int,bool> visited;
    createadj(adj,edges);
    for(int i=0;i<vertex;i++){
        if(!visited[i]){
            bfs(adj,visited,ans,i);
        }
```

```
    }
}


int main(){

int v=5;
 vector<pair<int, int>> edges = {{0, 1}, {0, 2}, {1, 3}, {2, 4}}; // Edges of
the graph

 vector<int>result=BFS(v,edges);
}
```

```cpp
#include <bits/stdc++.h>
using namespace std;
void dfs(int node, unordered_map<int, bool> &visited, unordered_map<int,
list<int>> &adj, vector<int> &component) {
    visited[node] = true;
    component.push_back(node);

    for (int neighbor : adj[node]) {
        if (!visited[neighbor]) {
            dfs(neighbor, visited, adj, component);
        }
    }
}

vector<vector<int>> depthFirstSearch(int v, int e, vector<vector<int>> &edges)
{
    unordered_map<int, list<int>> adj;

    // prepare adjlist
    for (int i = 0; i < edges.size(); i++) {

        int u = edges[i][0];
        int v = edges[i][1];
        adj[u].push_back(v);
        adj[v].push_back(u);
    }

    vector<vector<int>> ans;
    unordered_map<int, bool> visited;

    // for all nodes call the dfs if node not visited
    for (int i = 0; i < v; i++) {
        if (!visited[i]) {
            vector<int> component;
            dfs(i, visited, adj, component);
            ans.push_back(component);
        }
    }

    return ans;
}

int main() {
    int vertices, edges;
    cout << "Enter the number of vertices: ";
    cin >> vertices;
    cout << "Enter the number of edges: ";
    cin >> edges;
```

```
    vector<vector<int>> edgeList(edges, vector<int>(2));

    cout << "Enter the edges (format: u v):" << endl;
    for (int i = 0; i < edges; ++i) {
        cin >> edgeList[i][0] >> edgeList[i][1];
    }

   vector<vector<int>> components = depthFirstSearch(vertices, edges,
edgeList);

    cout << "Connected components in the graph:" << endl;
    for (const auto& component : components) {
        for (int node : component) {
            cout << node << " ";
        }
        cout << endl;
    }

    return 0;
}
```

```cpp
#include <bits/stdc++.h>
using namespace std;
//adjancy listtt
    unordered_map<int,list<int>>adj;
     void addedge(int u,int v,bool direction){
        //direction=0->undirected
        //direction =1->directed


        //create an edge from u to v
        adj[u].push_back(v);
        if(direction==0)
        adj[v].push_back(u);

     }

     void printadj(){


        for(auto i:adj){
            cout<<i.first<<"->",
            for(auto j:i.second){
                cout<<j<<",";
            }
        }
     }

     int main(){

        int n;
        cout<<"enter the number of the nodes"<<end;
        cin>>n;
        int m;
        cout<<"enter the number of edges"<<endl;
        cin>>m;

        graph g;

            for(int i=0;i<m;i++){
                int u,v;
                cin>>u>>v;
                //creating an undirected graph
                g.addedge(u,v,0);

            }
            //print the graph
            g.printadj();

            return 0;}
```

```c
//Breadth first traversal
#include<stdio.h>
#include<conio.h>
int a[20][20],q[20],visited[20],n,i,j,f=-1,r=0;
voidbfs(int v){
q[++r]=v;
visited[v]=1;
while(f<=r) {
for(i=1;i<=n;i++)
if(a[v][i] && !visited[i]){
visited[i]=1;
q[++r]=i;
 }
f++;
v=q[f];
}
}
void main(){
int v;
printf("\n Enter the number of vertices:");
scanf("%d",&n);
for(i=1;i<=n;i++){
 q[i]=0;
visited[i]=0;
}
printf("\n Enter graph data in matrix form:\n");
for(i=1;i<=n;i++)
for(j=1;j<=n;j++)
scanf("%d",&a[i][j]);
printf("\n Enter the starting vertex:");
scanf("%d",&v);
bfs(v);

printf("\n The node which are reachable are:\n");
for(i=1;i<=n;i++)
if(visited[i])
printf("%d\t",q[i]);
else
printf("\n Bfs is not possible");
}
//Checking whether a given graph is connected or not using DFS method
#include<stdio.h>
#include<conio.h>
int a[20][20],reach[20],n;
void dfs(int v){
int i; reach[v]=1;
for(i=1;i<=n;i++)
if(a[v][i] && !reach[i]) {
```

```
printf("\n %d->%d",v,i);
dfs(i);
}
}
void main(){
int i,j,count=0;
printf("\n Enter number of vertices:");
scanf("%d",&n);
for(i=1;i<=n;i++){
reach[i]=0;
for(j=1;j<=n;j++)
a[i][j]=0;
}
printf("\n Enter the adjacency matrix:\n");
for(i=1;i<=n;i++)
for(j=1;j<=n;j++)
scanf("%d",&a[i][j]);
dfs(1);
printf("\n");
for(i=1;i<=n;i++){
if(reach[i])
count++;
}
if(count==n)
printf("\n Graph is connected");
else
printf("\n Graph is not connected");
}
```

```cpp
#include <iostream>
#include <vector>

using namespace std;

int knapsack(int capacity, const vector<int>& weights, const vector<int>&
values, int n) {
    vector<vector<int>> dp(n + 1, vector<int>(capacity + 1, 0));

    for (int i = 1; i <= n; ++i) {
        for (int w = 1; w <= capacity; ++w) {
            if (weights[i - 1] <= w) {
                dp[i][w] = max(values[i - 1] + dp[i - 1][w - weights[i - 1]],
dp[i - 1][w]);
            } else {
                dp[i][w] = dp[i - 1][w];
            }
        }
    }

    return dp[n][capacity];
}

int main() {
    vector<int> values = {60, 100, 120};
    vector<int> weights = {10, 20, 30};
    int capacity = 50;
    int n = values.size();

    int maxValue = knapsack(capacity, weights, values, n);
    cout << "Maximum value that can be obtained: " << maxValue << endl;

    return 0;
}
```

```cpp
#include <iostream>

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

    int leftArr[n1], rightArr[n2];

    for (int i = 0; i < n1; i++)
        leftArr[i] = arr[left + i];
    for (int j = 0; j < n2; j++)
        rightArr[j] = arr[mid + 1 + j];

    int 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 iterativeMergeSort(int arr[], int n) {
    for (int currSize = 1; currSize <= n - 1; currSize = 2 * currSize) {
        for (int leftStart = 0; leftStart < n - 1; leftStart += 2 * currSize)
{
            int mid = leftStart + currSize - 1;
            int rightEnd = std::min(leftStart + 2 * currSize - 1, n - 1);

            merge(arr, leftStart, mid, rightEnd);
        }
```

```
    }
}

void printArray(const int arr[], int size) {
    for (int i = 0; i < size; i++) {
        std::cout << arr[i] << " ";
    }
    std::cout << std::endl;
}

int main() {
    int arr[] = {12, 11, 13, 5, 6, 7};
    int size = sizeof(arr) / sizeof(arr[0]);

    std::cout << "Original array: ";
    printArray(arr, size);

    iterativeMergeSort(arr, size);

    std::cout << "Sorted array: ";
    printArray(arr, size);

    return 0;
}
```

```cpp
#include <iostream>
using namespace std;

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

    int leftArr[n1], rightArr[n2];

    for (int i = 0; i < n1; i++)
        leftArr[i] = arr[left + i];
    for (int j = 0; j < n2; j++)
        rightArr[j] = arr[mid + 1 + j];

    int 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 recursiveMergeSort(int arr[], int left, int right) {
    if (left < right) {

        int mid = left + (right - left) / 2;

        recursiveMergeSort(arr, left, mid);
        recursiveMergeSort(arr, mid + 1, right);
```

```
        merge(arr, left, mid, right);
    }
}

void printArray(const int arr[], int size) {
    for (int i = 0; i < size; i++) {
        std::cout << arr[i] << " ";
    }
    std::cout << std::endl;
}

int main() {
    int arr[] = {12, 11, 13, 5, 6, 7};
    int size = sizeof(arr) / sizeof(arr[0]);
    std::cout << "Original array: ";
    printArray(arr, size);
    recursiveMergeSort(arr, 0, size - 1);
     std::cout << "Sorted array: ";
    printArray(arr, size);

    return 0;
}
```

```
#include<stdio.h>
#include<stdlib.h>
inti,j,k,a,b,u,v,n,ne=1;
intmin,mincost=0,cost[9][9],parent[9];
int find(int);
intuni(int,int);
void main() {
printf("\n Implementation of Kruskal's algorithm\n\n");
printf("\nEnter the no. of vertices\n");
scanf("%d",&n);
printf("\nEnter the cost adjacency matrix\n");
for(i=1;i<=n;i++){
for(j=1;j<=n;j++) {
scanf("%d",&cost[i][j]);
30 | P a g e
if(cost[i][j]==0)
cost[i][j]=999;
}
}
printf("\nThe edges of Minimum Cost Spanning Tree are\n\n");
while(ne<n){
for(i=1,min=999;i<=n;i++) {
for(j=1;j<=n;j++){
if(cost[i][j]<min){
min=cost[i][j];
a=u=i;
b=v=j; }}}
u=find(u);
v=find(v);
if(uni(u,v)){
printf("\n%d edge (%d,%d) =%d\n",ne++,a,b,min);
mincost +=min;
}
cost[a][b]=cost[b][a]=999;
 }
printf("\n\tMinimum cost = %d\n",mincost);
}
int find(int i){
while(parent[i])
i=parent[i];
return i;
}
intuni(inti,int j){
if(i!=j) {
parent[j]=i;
return 1;
}
return 0;}
```

```cpp
#include<bits/stdc++.h>
using namespace std;
void swap(int *x,int *y){
    int temp=*x;
    *x=*y;
    *y=temp;
}

int partition(int a[],int l,int h){
    int pivot=a[l];
    int i=l,j=h;
    do{

        do{i++;}while(a[i]<=pivot);
        do{j--;}while(a[i]>pivot);

        if(i<j)
        swap(&a[i],a[j]);

    }while(i<j);

    swap(&a[l],&A[j]);
    return j;
}

void quicksort(int a[],int l,int h){
    int j;
    while(l<h){

        j=partition(a,l,h);
        quicksort(a,l,j);
        qiucksort(a,j+1,h);

    }
}
int main(){
    int n;
    cin>>n;
    int a[n];
    for(int i=0;i<n;i++)
    cin>>a[i];

    quicksort(a,0,n-1);
    for(int i=0;i<n;i++)
    cout<<a[i];
}
```

```cpp
#include <iostream>
#include <cstdlib> // Include for rand() function
using namespace std;

void swap(int *x, int *y) {
    int temp = *x;
    *x = *y;
    *y = temp;
}

int partition(int a[], int l, int h) {
    int pivot = a[l];
    int i = l, j = h;
    do {
        do { i++; } while (a[i] <= pivot);
        do { j--; } while (a[j] > pivot);
        if (i < j)
            swap(&a[i], &a[j]);
    } while (i < j);

    swap(&a[l], &a[j]);
    return j;
}

void quicksort(int a[], int l, int h) {
    int j;
    if (l < h) {
        j = partition(a, l, h);
        quicksort(a, l, j);
        quicksort(a, j + 1, h);
    }
}

int main() {
    int n;
    cout << "Enter the size of the array: ";
    cin >> n;
    int a[n];

    // Generating random elements for the array
    for (int i = 0; i < n; i++)
        a[i] = rand() % 1000; // Generating random numbers between 0 to 999

    cout << "Input Array: ";
    for (int i = 0; i < n; i++)
        cout << a[i] << " ";

    quicksort(a, 0, n - 1);
```

```
    cout << "\nSorted Array: ";
    for (int i = 0; i < n; i++)
        cout << a[i] << " ";

    return 0;
}
```

```cpp
#include <iostream>
#include <fstream> // For file handling
using namespace std;

void swap(int *x, int *y) {
    int temp = *x;
    *x = *y;
    *y = temp;
}

int partition(int a[], int l, int h) {
    int pivot = a[l];
    int i = l, j = h;
    do {
        do { i++; } while (a[i] <= pivot);
        do { j--; } while (a[j] > pivot);
        if (i < j)
            swap(&a[i], &a[j]);
    } while (i < j);

    swap(&a[l], &a[j]);
    return j;
}

void quicksort(int a[], int l, int h) {
    int j;
    if (l < h) {
        j = partition(a, l, h);
        quicksort(a, l, j);
        quicksort(a, j + 1, h);
    }
}

int main() {
    ifstream inputFile("input.txt"); // Open input file
    if (!inputFile.is_open()) {
        cout << "Unable to open file";
        return 1;
    }

    int n;
    inputFile >> n; // Read the size of the array
    int a[n];

    // Read elements from the file into the array
    for (int i = 0; i < n; i++)
        inputFile >> a[i];
```

```
    inputFile.close(); // Close the file

    // Sorting the array
    quicksort(a, 0, n - 1);

    // Output the sorted array
    for (int i = 0; i < n; i++)
        cout << a[i] << " ";

    return 0;
}
```

```c
#include<stdio.h>
#define infinity 999
void dij(int n, int v,int cost[20][20], int dist[]){

int i,u,count,w,flag[20],min;
for(i=1;i<=n;i++)
flag[i]=0, dist[i]=cost[v][i];
count=2;
while(count<=n){
min=99;
for(w=1;w<=n;w++)
if(dist[w]<min && !flag[w]) {
min=dist[w];
u=w;
}
flag[u]=1;
count++;
for(w=1;w<=n;w++)
if((dist[u]+cost[u][w]<dist[w]) && !flag[w])
dist[w]=dist[u]+cost[u][w];
}
}
int main(){
int n,v,i,j,cost[20][20],dist[20];
printf("enter the number of nodes:");
scanf("%d",&n);
printf("\n enter the cost matrix:\n");
for(i=1;i<=n;i++)
for(j=1;j<=n;j++){
scanf("%d",&cost[i][j]);
if(cost[i][j] == 0)
cost[i][j]=infinity;
}
printf("\n enter the source matrix:");
scanf("%d",&v);
dij(n,v,cost,dist);
printf("\n shortest path : \n");
for(i=1;i<=n;i++)
if(i!=v)
printf("%d->%d,cost=%d\n",v,i,dist[i]);
}
```

```
#include<stdio.h>
#define TRUE 1
#define FALSE 0
int inc[50],w[50],sum,n;
voidsumset(int ,int ,int);
int promising(inti,intwt,int total) {
return (((wt+total)>=sum)&&((wt==sum)||(wt+w[i+1]<=sum)));
}
void main() {
inti,j,n,temp,total=0;
printf("\n Enter how many numbers: ");
scanf("%d",&n);
printf("\n Enter %d numbers : ",n);
for (i=0;i<n;i++) {
scanf("%d",&w[i]);
total+=w[i];
}
printf("\n Input the sum value to create sub set: ");
scanf("%d",&sum);
for (i=0;i<=n;i++)
for (j=0;j<n-1;j++)

if(w[j]>w[j+1]) {
temp=w[j];
w[j]=w[j+1];
w[j+1]=temp;
}
printf("\n The given %d numbers in ascending order: ",n);
for (i=0;i<n;i++)
printf("%3d",w[i]);
if((total<sum))
printf("\n Subset construction is not possible");
else{
for (i=0;i<n;i++)
inc[i]=0;
printf("\n The solution using backtracking is:\n");
sumset(-1,0,total);
}
}
voidsumset(inti,intwt,int total){
int j;
if(promising(i,wt,total)) {
if(wt==sum){
printf("\n{");
for (j=0;j<=i;j++)
if(inc[j])
printf("%3d",w[j]);
printf(" }\n");
```

```
} else {
inc[i+1]=TRUE;
sumset(i+1,wt+w[i+1],total-w[i+1]);
inc[i+1]=FALSE;
sumset(i+1,wt,total-w[i+1]);
 }
}
}
```

```cpp
#include <bits/stdc++.h>

using namespace std;

void topologicalSortDFS(int node, unordered_map<int, bool> &visited,
stack<int> &s, unordered_map<int, list<int>> &adj) {
    visited[node] = true;
    for (int neighbor : adj[node]) {
        if (!visited[neighbor]) {
            topologicalSortDFS(neighbor, visited, s, adj);
        }
    }
    s.push(node);
}

vector<int> topologicalSort(vector<vector<int>>& edges, int v, int e) {
    unordered_map<int, list<int>> adj;
    for (int i = 0; i < e; i++) {
        int u = edges[i][0];
        int v = edges[i][1];

        adj[u].push_back(v);
    }

    unordered_map<int, bool> visited;
    stack<int> s;
    for (int i = 0; i < v; i++) {
        if (!visited[i]) {
            topologicalSortDFS(i, visited, s, adj);

        }
    }

    vector<int> ans;
    while (!s.empty()) {
        ans.push_back(s.top());
        s.pop();
    }

    return ans;
}

int main() {
    // Example usage
    int v = 6, e = 6;
    vector<vector<int>> edges = {{5, 2}, {5, 0}, {4, 0}, {4, 1}, {2, 3}, {3,
1}};
```

```
    vector<int> result = topologicalSort(edges, v, e);

    cout << "Topological ordering: ";
    for (int node : result) {
        cout << node << " ";
    }
    cout << endl;

    return 0;
}
```

```cpp
#include <iostream>
#include <vector>

using namespace std;

vector<vector<int>> transitiveClosure(const vector<vector<int>>& graph) {
    int numVertices = graph.size();
    vector<vector<int>> closure = graph;

    for (int k = 0; k < numVertices; ++k) {
        for (int i = 0; i < numVertices; ++i) {
            for (int j = 0; j < numVertices; ++j) {
                closure[i][j] = closure[i][j] || (closure[i][k] &&
closure[k][j]);
            }
        }
    }

    return closure;
}

void printMatrix(const vector<vector<int>>& matrix) {
    for (const auto& row : matrix) {
        for (int value : row) {
            cout << value << " ";
        }
        cout << endl;
    }
}

int main() {
    // Example graph represented as an adjacency matrix
    vector<vector<int>> graph = {
        {1, 1, 0, 1},
        {0, 1, 1, 0},
        {0, 0, 1, 1},
        {0, 0, 0, 1}
    };

    vector<vector<int>> result = transitiveClosure(graph);

    cout << "Transitive Closure:" << endl;
    printMatrix(result);

    return 0;
}
```

```
#include<stdio.h>
ints,c[100][100],ver;
float optimum=999,sum;
/* function to swap array elements */
void swap(int v[], int i, int j) {
int t;
t = v[i];
v[i] = v[j];
v[j] = t;
}
/* recursive function to generate permutations */
voidbrute_force(int v[], int n, int i) {
// this function generates the permutations of the array from element i to
element n-1
int j,sum1,k;
//if we are at the end of the array, we have one permutation
if (i == n) {
if(v[0]==s) {
for (j=0; j<n; j++)
printf ("%d ", v[j]);
sum1=0;

for( k=0;k<n-1;k++) {
sum1=sum1+c[v[k]][v[k+1]];
}
sum1=sum1+c[v[n-1]][s];
printf("sum = %d\n",sum1);
if (sum1<optimum)
optimum=sum1;
}
}
else
// recursively explore the permutations starting at index i going through
index n-1*/
for (j=i; j<n; j++) { /* try the array with i and j switched */
swap (v, i, j);
brute_force (v, n, i+1);
/* swap them back the way they were */
swap (v, i, j);
}
}
voidnearest_neighbour(intver) {
intmin,p,i,j,vis[20],from;
for(i=1;i<=ver;i++)
vis[i]=0;
vis[s]=1;
from=s;
sum=0;
```

```
for(j=1;j<ver;j++) {
min=999;
for(i=1;i<=ver;i++)
if(vis[i] !=1 &&c[from][i]<min && c[from][i] !=0 ) {
min= c[from][i];
p=i;
}
vis[p]=1;
from=p;
sum=sum+min;
}
sum=sum+c[from][s];
}
void main () {
intver,v[100],i,j;
printf("Enter n : ");
scanf("%d",&ver);
for (i=0; i<ver; i++)
v[i] = i+1;
printf("Enter cost matrix\n");
for(i=1;i<=ver;i++)
for(j=1;j<=ver;j++)
scanf("%d",&c[i][j]);
printf("\nEnter source : ");
scanf("%d",&s);
47 | P a g e
brute_force (v, ver, 0);
printf("\nOptimum solution with brute force technique is=%f\n",optimum);
nearest_neighbour(ver);
printf("\nSolution with nearest neighbour technique is=%f\n",sum);
printf("The approximation val is=%f",((sum/optimum)-1)*100);
printf(" % ");
}
```