**Graph- DFS (Depth First Search)**

// DFS (Depth First Search)

#include<bits/stdc++.h>

using namespace std;

const int N = 1e5+10;

vector<int>graph[N];

bool visited[N];

void dfs(int src){

visited[src] = true;

cout << src << " ";

for(auto child:graph[src]){

if(visited[child] == true) continue;

dfs(child);

}

}

int main(){

int n,m; cin >> n >> m;

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

int v1,v2; cin >> v1 >> v2;

graph[v1].push\_back(v2);

graph[v2].push\_back(v1);

}

dfs(1);

}

**Graph – BFS (Breadth First Search)**

#include<bits/stdc++.h>

using namespace std;

const int N = 1e5+10;

int visited[N];

vector<int>graph[N];

void bfs(int src){

queue<int> q;

q.push(src);

visited[src] = 1;

while(!q.empty()) {

int cur = q.front();

cout << cur << " ";

q.pop();

for(auto child:graph[cur]){

if(!visited[child]){

q.push(child);

visited[child] = 1;

}

}

}

}

int main(){

int n; cin >> n;

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

int v1, v2; cin >> v1 >> v2;

graph[v1].push\_back(v2);

graph[v2].push\_back(v1);

}

bfs(1);

}

**Tree (insert, delete, traversal)**

// Tree (insert, delete, traversal).cpp

#include <bits/stdc++.h>

using namespace std;

struct Node{

int data;

Node \*left;

Node \*right;

};

struct Node \*createNode(int data){

Node \*newNode = new Node;

newNode->data = data;

newNode->left = NULL;

newNode->right = NULL;

return newNode;

}

**// Insert node......**

struct Node \*insertNode(Node \*ptr, int data){

if (ptr == NULL)

ptr = createNode(data);

else if (ptr->data >= data)

ptr->left = insertNode(ptr->left, data);

else

ptr->right = insertNode(ptr->right, data);

return ptr;

}

**// Pre-Order traversal......**

void preOrder(Node \*ptr){

if (ptr != NULL){

cout << ptr->data << " ";

preOrder(ptr->left);

preOrder(ptr->right);

}

}

**// Post-Order traversal......**

void postOrder(Node \*ptr){

if (ptr != NULL){

postOrder(ptr->left);

postOrder(ptr->right);

cout << ptr->data << " ";

}

}

**// In-Order traversal......**

void inOrder(Node \*ptr){

if (ptr != NULL){

inOrder(ptr->left);

cout << ptr->data << " ";

inOrder(ptr->right);

}

}

void display (Node \*root){

cout << "Current list!!\n";

cout << "Pre Order: ";

preOrder(root);

cout << endl;

cout << "In Order: ";

inOrder(root);

cout << endl;

cout << "Post Order: ";

postOrder(root);

cout << endl;

}

int main(){

Node \*root = NULL;

while (1){

system("cls");

display(root);

cout << "\nEnter I for insert element!\n";

cout << "Enter any for exit!\n";

cout << "Enter your choice: ";

char ch;

cin >> ch;

if (ch == 'i' || ch == 'I'){

system("cls");

display(root);

cout << "Enter element for insert: ";

int element;

cin >> element;

root = insertNode(root, element);

}

else

break;

}

}

**DFS- Depth First Search for TREE**

// DFS- Depth First Search for TREE

#include<bits/stdc++.h>

using namespace std;

const int N = 1e5+10;

vector<int>tree[N];

void dfs(int src,int parent){

cout << src << " ";

for(auto child:tree[src]){

if(child==parent) continue;

dfs(child,src);

}

}

int main(){

int n; cin >> n;

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

int v1,v2; cin >> v1 >> v2;

tree[v1].push\_back(v2);

tree[v2].push\_back(v1);

}

dfs(1,0);

}

**Kruskal's Algorithm - Minimum Spanning Tree**

//Kruskal's Algorithm: Minimum Spanning Tree

#include<bits/stdc++.h>

using namespace std;

const int N = 1e5+10;

int parent[N],sz[N];

void make(int v){

    parent[v] = v;

    sz[v] = 1;

}

int find(int v){

    if(parent[v] == v) return v;

    return parent[v] = find(parent[v]);

}

void Union(int a,int b){

    a = find(a);

    b = find(b);

    if(a != b){

        if(sz[a] < sz[b]) swap(a,b);

    }

    parent[b] = a;

    sz[a] += sz[b];

}

int main(){

    int n,m;    cin >> n >> m;

    vector<pair<int,pair<int,int>>> edges;

    for(int i=0;i<m;i++){

        int v1,v2,wt;

        cin >> v1 >> v2 >> wt;

        edges.push\_back({wt,{v1,v2}});

    }

    sort(edges.begin(), edges.end());

    for(int i=1;i<=n;i++){

        make(i);

    }

    int total\_cost = 0;

    for(auto it:edges){

        int wt = it.first;

        int v1 = it.second.first;

        int v2 = it.second.second;

        if(find(v1) == find(v2)) continue;

        Union(v1,v2);

        total\_cost +=wt;

        cout << v1 << " " << v2 << endl;

    }

    cout << "Total Cost: " << total\_cost << endl;

}