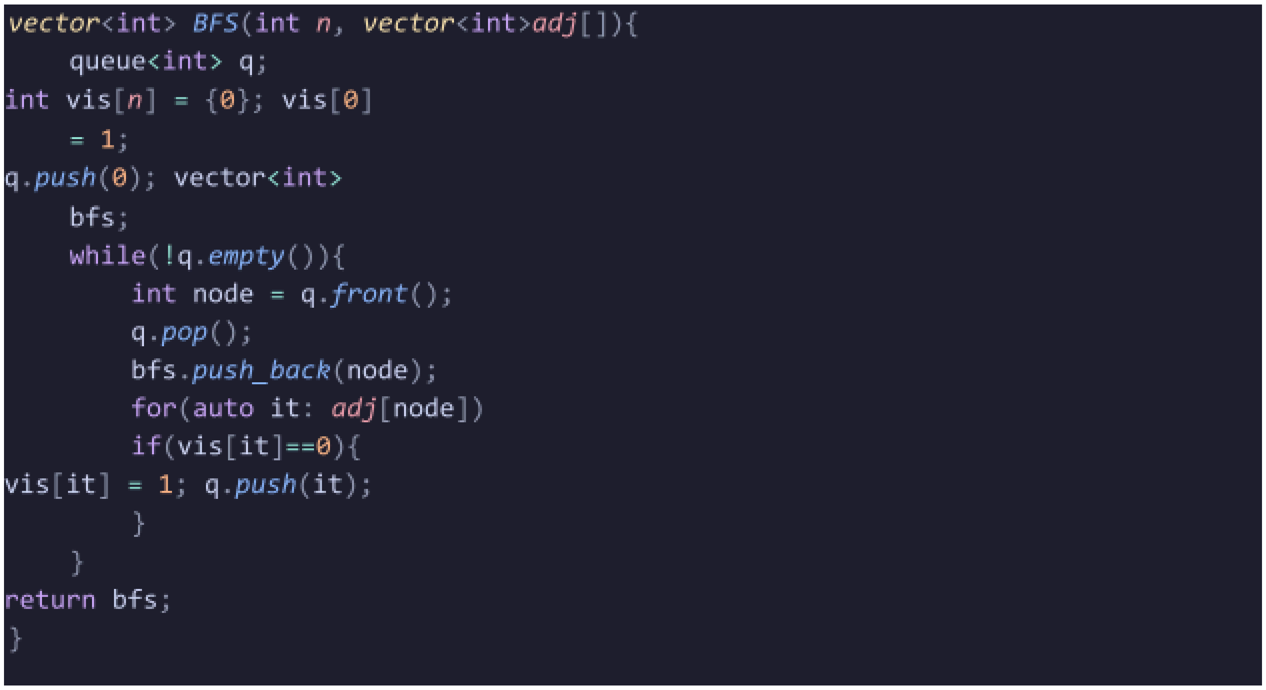
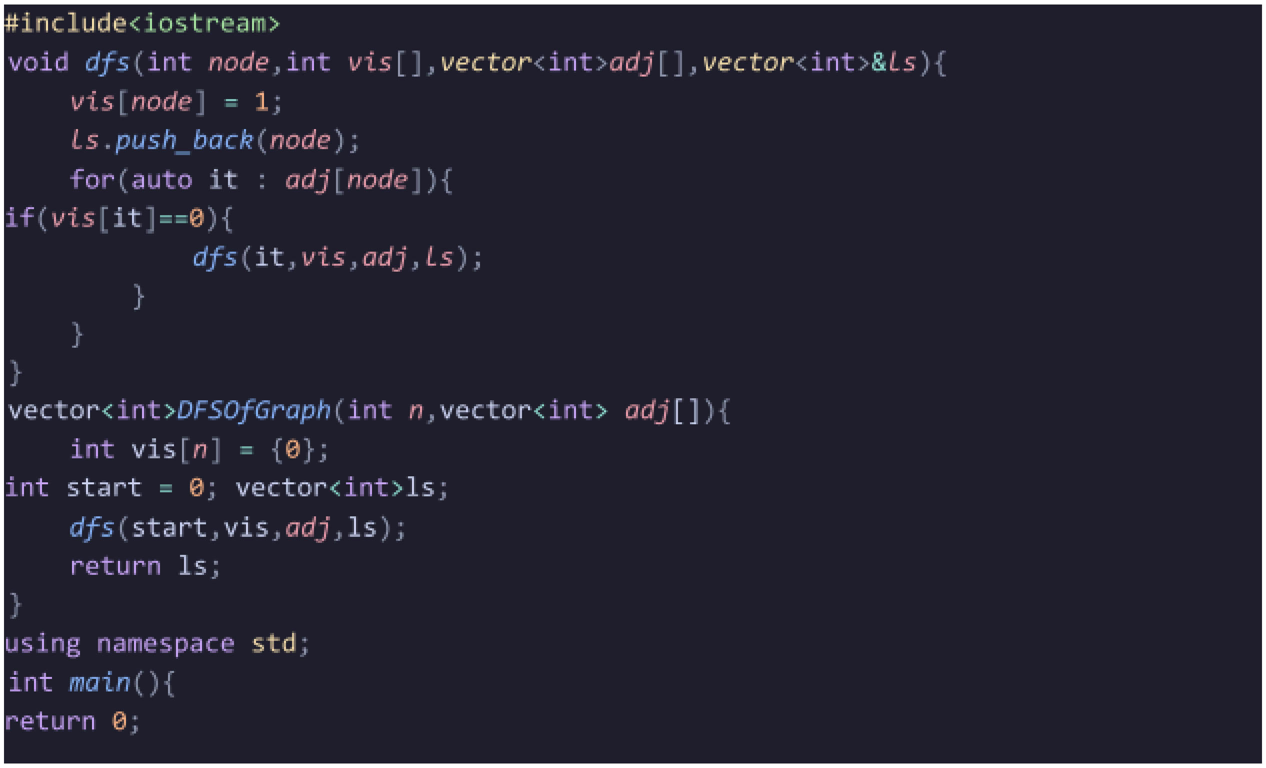
Data Structures (UCS301) Assignment

NAME-Suneet Arora **ROLL NO-1024030174**



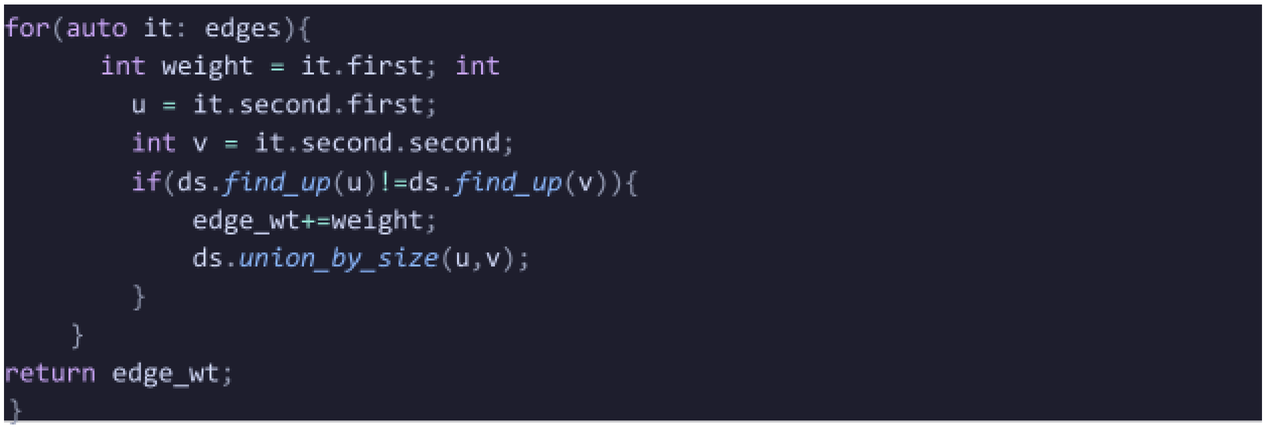
```
vector<int> BFS(int n, vector<int>adj[]){
    queue<int> q;
int vis[n] = {0}; vis[0]
    = 1;
q.push(0); vector<int>
    bfs;
    while(!q.empty()){
        int node = q.front();
        q.pop();
        bfs.push_back(node);
        for(auto it: adj[node])
        if(vis[it]==0){
vis[it] = 1; q.push(it);
        }
    }
return bfs;
}
```



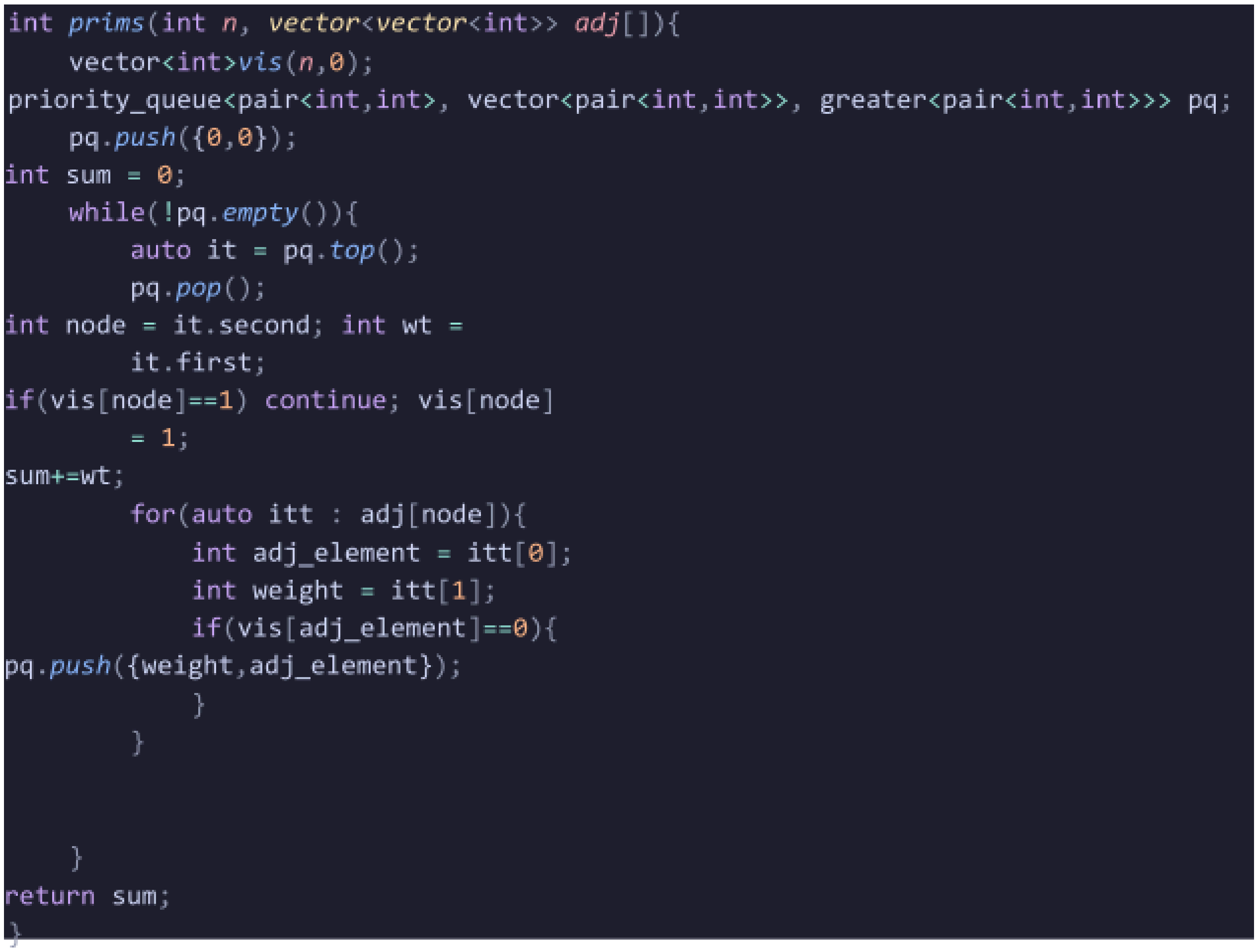
```
#include<iostream>
void dfs(int node,int vis[],vector<int>adj[],vector<int>&ls){
    vis[node] = 1;
    ls.push_back(node);
    for(auto it : adj[node]){
if(vis[it]==0){
            dfs(it,vis,adj,ls);
        }
    }
}
vector<int>DFSOfGraph(int n,vector<int> adj[]){
    int vis[n] = {0};
int start = 0; vector<int>ls;
    dfs(start,vis,adj,ls);
    return ls;
}
using namespace std;
int main(){
return 0;
}
```

```
vector<int> dijkstra(int n,vector<vector<int>> adj[],int s){
    priority_queue<pair<int,int>,vector<pair<int,int>>,greater<pair<int,int>>>
pq;
vector<int>dist(n); for(int
    i=0;i<n;i++){
dist[i] = INT_MAX;
    }
dist[s] = 0;
    pq.push({0,s});
    while(!pq.empty()){
int dis = pq.top().first; int node =
        pq.top().second; pq.pop();

        for(auto it: adj[node]){
            int wt = it[1];
int adjnode = it[0];
            if(dis + wt<dist[adjnode]){
                dist[adjnode] = dis+wt;
                pq.push({dist[adjnode],adjnode});
            }
        }
    }
return dist;
}
```

```
class Disjoint_set{
vector<int> rank,size,parent;
    public:
    Disjoint_set(int n){
        size.resize(n+1,1);
        rank.resize(n+1,0);
        parent.resize(n+1);
        for(int i=0;i<=n;i++){
parent[i] = i;
        }
    }
    int find_up(int node){
        if(parent[node] == node) return node; return
        parent[node] = find_up(parent[node]);
    }
    void union_by_rank(int u,int v){
        int ulp_u = find_up(u);
        int ulp_v = find_up(v);
        if(ulp_u==ulp_v) return;
        if(rank[ulp_u]>rank[ulp_v]){
            parent[ulp_v] = ulp_u;
        }
        else if(rank[ulp_v]>rank[ulp_u]){
            parent[ulp_u] = ulp_v;
        }
        else{
            parent[ulp_v] = ulp_u;
            rank[ulp_u]++;
        }
    }
    void union_by_size(int u,int v){
        int ulp_u = find_up(u);
        int ulp_v = find_up(v);
        if(ulp_u==ulp_v) return;
        if(size[ulp_u]<size[ulp_v]){
            parent[ulp_u] = ulp_v;
            size[ulp_v]+=size[ulp_u];
        }

        else{
            parent[ulp_v] = ulp_u;
            size[ulp_u]+=size[ulp_v];
        }
    }
};
int kruskal(int n,vector<vector<int>> adj[]){
    vector<pair<int,pair<int,int>>> edges;
    for(int i=0;i<n;i++){
        for(auto it : adj[i]){
            int adjnode = it[0];
            int wt = it[1];
            int node = i;
            edges.push_back({wt,{node,adjnode}});
        }
    }
    sort(edges.begin(),edges.end());
    int edge_wt = 0;
    Disjoint_set ds(n);
```



```
for(auto it: edges){
      int weight = it.first; int
        u = it.second.first;
        int v = it.second.second;
        if(ds.find_up(u)!=ds.find_up(v)){
            edge_wt+=weight;
            ds.union_by_size(u,v);
        }
    }
return edge_wt;
}
```



```
int prims(int n, vector<vector<int>> adj[]){
    vector<int>vis(n,0);
priority_queue<pair<int,int>, vector<pair<int,int>>, greater<pair<int,int>>> pq;
    pq.push({0,0});
int sum = 0;
    while(!pq.empty()){
        auto it = pq.top();
        pq.pop();
int node = it.second; int wt =
        it.first;
if(vis[node]==1) continue; vis[node]
        = 1;
sum+=wt;
        for(auto itt : adj[node]){
            int adj_element = itt[0];
            int weight = itt[1];
            if(vis[adj_element]==0){
pq.push({weight,adj_element});
            }
        }



    }
return sum;
}
```