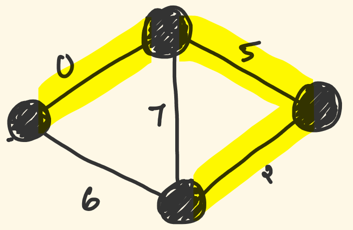
Minimum spanning trees

Sunday, October 29, 2023 6:40 AM

- subset of edges which connects all vertices together (without cycles) while minimizing the total cost.

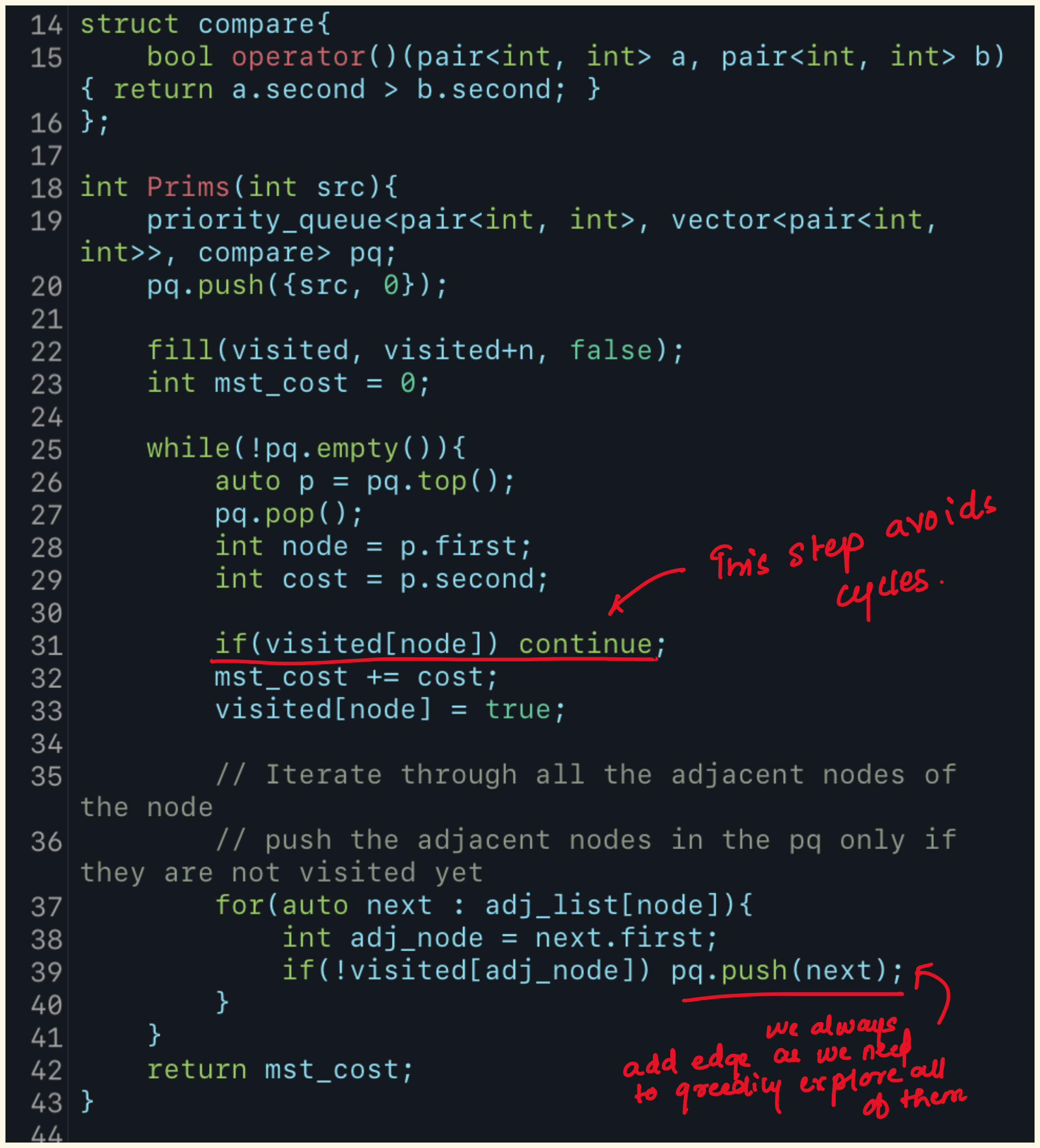


Prim's MST Algorithm

- Greedy algo that works well on dense graphs.
- Performs better than other algos on dense graphs.
- However, in case of a disconnected graph the algo has to be run on every single component.

Must be run on each component

-2 implementations O(E× log(E)) & → O(E× log(V)) faster



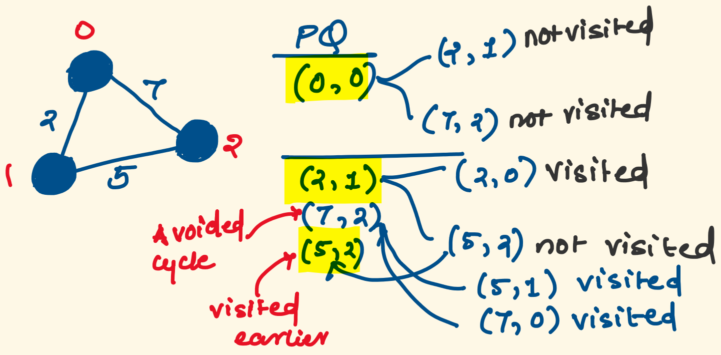
```
struct compare{
    bool operator()(pair<int, int> a, pair<int, int> b) { return a.second > b.second; }
};

int Prims(int src){
    priority_queue<pair<int, int>, vector<pair<int, int>>, compare> pq;
    pq.push({src, 0});

    fill(visited, visited+n, false);
    int mst_cost = 0;

    while(!pq.empty()){
        auto p = pq.top();
        pq.pop();
        int node = p.first;
        int cost = p.second;

        if(visited[node]) continue;
        mst_cost += cost;
        visited[node] = true;

        // Iterate through all the adjacent nodes of the node
        // push the adjacent nodes in the pq only if they are not visited yet
        for(auto next : adj_list[node]){
            int adj_node = next.first;
            if(!visited[adj_node]) pq.push(next);
        }
    }
    return mst_cost;
}
```

This step avoids cycles.

we always add edge as we need to greedily explore all of them

The reason we check visited twice is that we are adding every single edge to the pq.

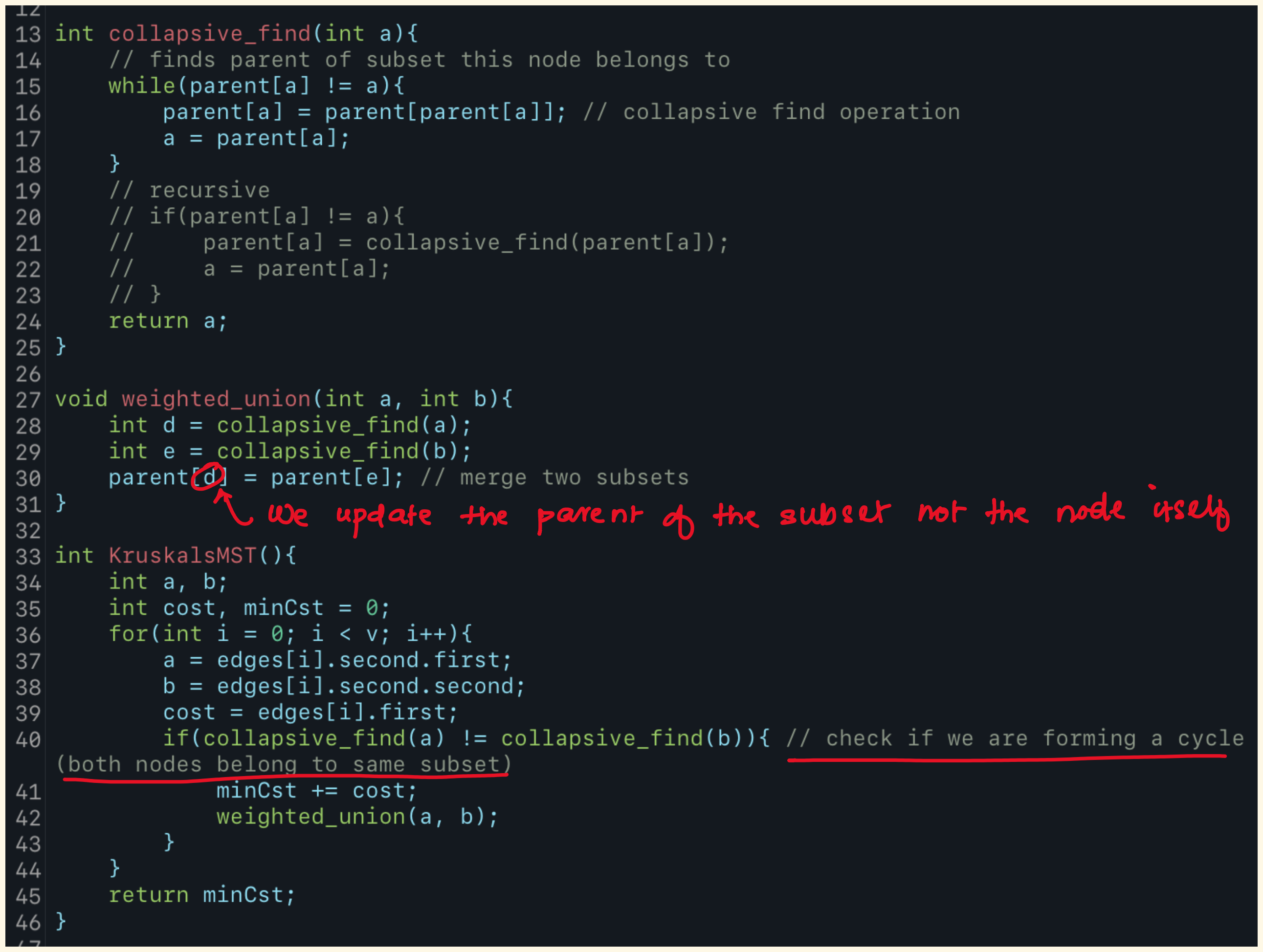
∴ A node might hv already been visited with a smaller cost.
- We avoid including the other edge with larger cost.



- We store all the edges to unvisited nodes in the priority queue.
- Priority queue will always return the min edge.

Kruskal's MST algorithm

- Sort all edges.
- Take edge with min cost.
- Repeat while discarding cycles.



```
int collapsive_find(int a){
    // finds parent of subset this node belongs to
    while(parent[a] != a){
        parent[a] = parent[parent[a]]; // collapsive find operation
        a = parent[a];
    }
    // recursive
    // if(parent[a] != a){
    //     parent[a] = collapsive_find(parent[a]);
    //     a = parent[a];
    // }
    return a;
}

void weighted_union(int a, int b){
    int d = collapsive_find(a);
    int e = collapsive_find(b);
    parent[d] = parent[e]; // merge two subsets
}

int KruskalsMST(){
    int a, b;
    int cost, minCst = 0;
    for(int i = 0; i < v; i++){
        a = edges[i].second.first;
        b = edges[i].second.second;
        cost = edges[i].first;
        if(collapsive_find(a) != collapsive_find(b)){ // check if we are forming a cycle (both nodes belong to same subset)
            minCst += cost;
            weighted_union(a, b);
        }
    }
    return minCst;
}
```

We update the parent of the subset not the node itself