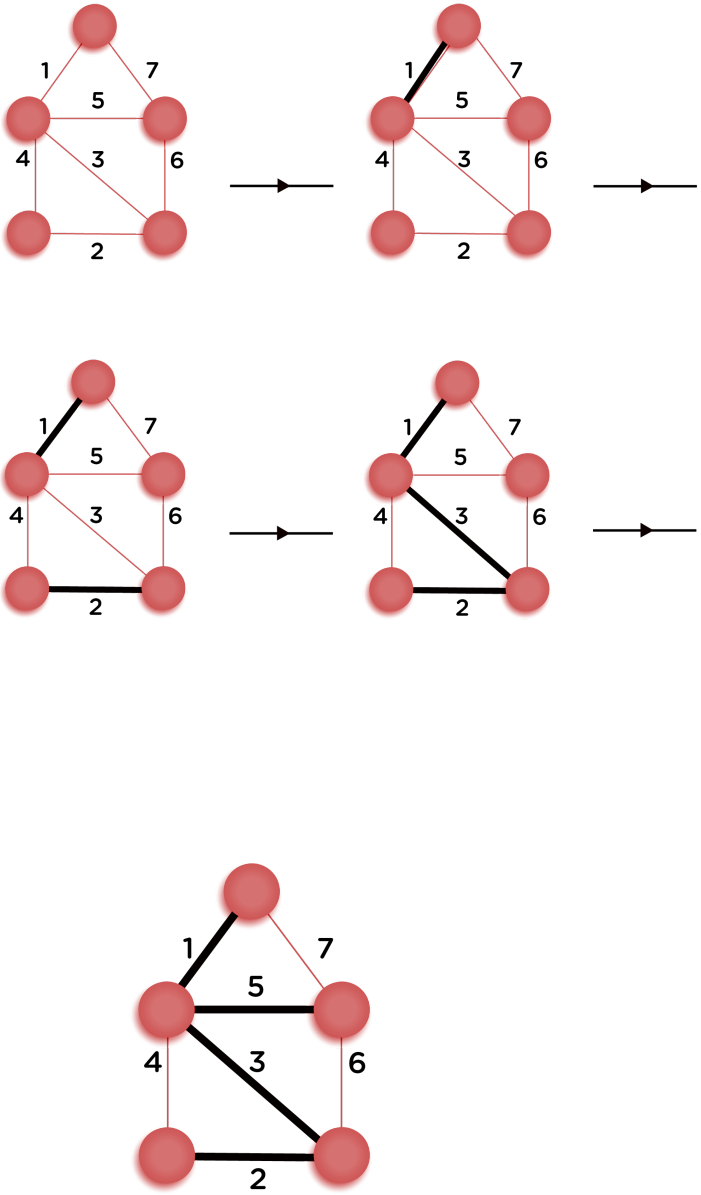
Kruskal's Algorithm

This algorithm is used to find MST for the given graph. It builds the spanning tree by adding edges one at a time. We start by picking the edge with minimum weight, adding that edge into the MST, and increasing the count of edges in the spanning tree by one. Now, we will be picking the minimum weighted edge by excluding the already chosen ones and correspondingly increasing the count. While choosing the edge, we will also make sure that the graph remains acyclic after including the same. This process will continue until the count of edges in the MST reaches *n-1*. Ultimately, the graph obtained will be MST.

Refer to the example below for a better understanding of the same.



This is the final MST obtained using Kruskal's algorithm. It can be checked manually that the final graph is the MST for the given graph.

Cycle Detection

While inserting a new edge in the MST, we have to check if introducing that edge makes the MST cyclic or not. If not, then we can include that edge, otherwise not.

Now, let's figure out a way to detect the cycle in a graph. The following are the possible cases:

- By including an edge between nodes A and B, if both nodes A and B are not present in the graph, then it is safe to include that edge as including it, will not bring a cycle to the graph.
- Out of two vertices, if any one of them has not been visited (or not present in the MST), then that vertex can also be included in the MST.
- If both the vertices are already present in the graph, they can introduce a cycle in the MST. It means we can't use this method to detect the presence of the cycle.

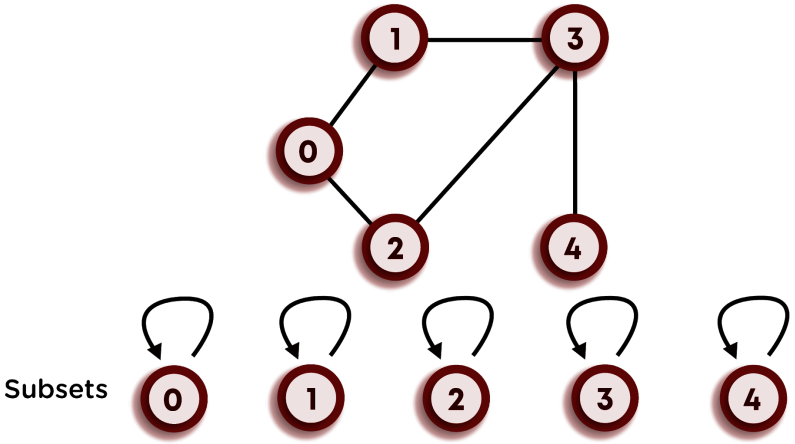
Union-Find Algorithm:

Before adding any edge to the graph, we will check if the two vertices of the edge lie in the same component of MST or not, in other words, we are checking whether the addition of an edge will lead to a cycle or not. If not, then it is safe to add that edge to the MST.

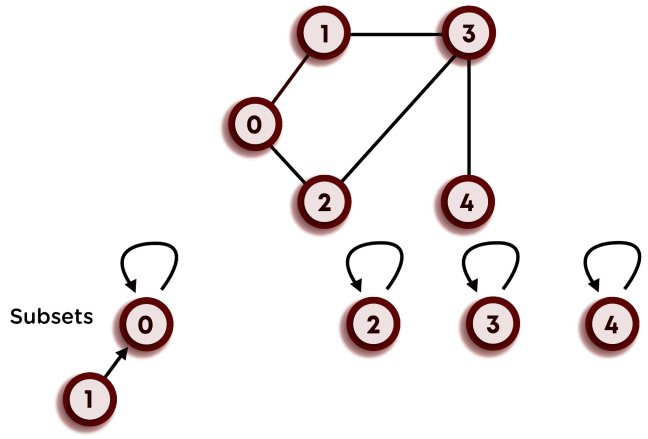
Following the steps of the algorithm

- We will assume that initially, the total number of disjoint sets is equal to the number of vertices in the graph starting from 0 to n-1.
- We will maintain a parent array specifying the parent vertex of each of the vertex of the graph. Initially, as each vertex belongs to a different disjoint set (connected component), hence each vertex will be its parent.
- Now, before inserting any edge into the MST, we will check the parent of the vertices. If their parent vertices are equal, they belong to the same connected component; hence it is unsafe to add that edge.
- Otherwise, we can add that edge into the MST, and simultaneously update the parent array so that they belong to the same component(Refer to the code on how to do so).

  Look at the following example, for better understanding:



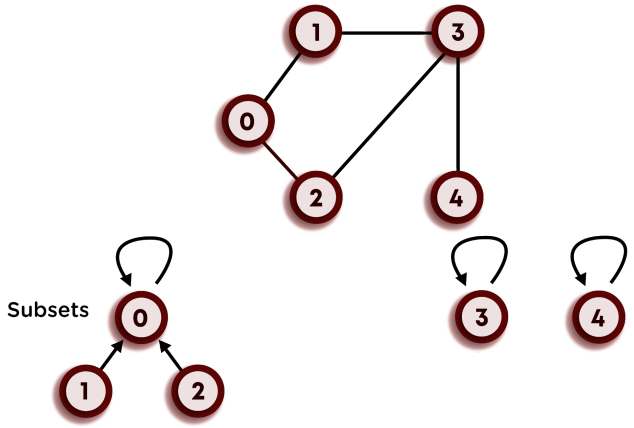
Initially all parent pointers are pointing to self
means only one element in each subset



Edge 0-1

Find: 0 belongs to subset 0 and 1 belongs to subset 1 so they are in different subsets.

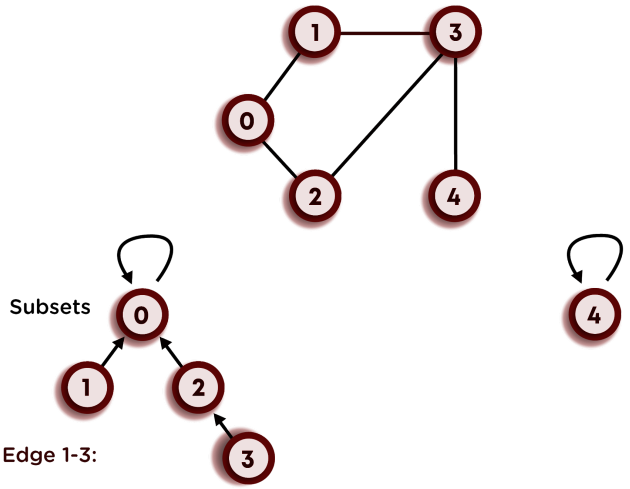
Union: Make 0 as the parent of 1, Updated set is {0,1}. 0 is the set representative since 0 is parent for itself.



Edge 0-2:

Find: 0 belongs to subset 0 and 2 belongs to subset 2 so they are in different subsets.

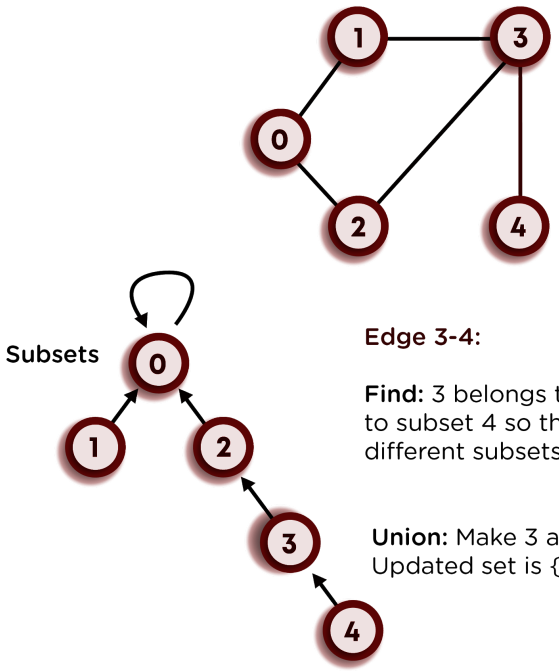
Union: Make 0 as the parent of 2, Updated set is {0,1,2}. 0 is the set representative since 0 is parent for itself.



Edge 1-3:

Find: 1 belongs to subset 0 and 3 belongs to subset 3 so they are in different subsets.

Union: Make 1 as the parent of 3, Updated set is {0,1,2,3}. 0 is the set representative since 0 is parent for itself.



Edge 3-4:

Find: 3 belongs to subset 0 and 4 belongs to subset 4 so they are in different subsets.

Union: Make 3 as the parent of 4, Updated set is {0,1,2,3,4}.

Note: While finding the parent of the vertex, we will be finding the topmost parent (Oldest Ancestor). For example: suppose, the vertex 0 and the vertex 1 were connected, where the parent of 0 is 1, and the parent of 1 is 1. Now, while determining the parent of the vertex 0, we will visit the parent array and check the vertex at index 0. In our case, it is 1. Now we will go to index 1 and check the parent of index 1, which is also 1. Hence, we can't go any further as the index is the parent of itself. This way, we will be determining the parent of any vertex.

The time complexity of the union-find algorithm becomes O(V) for each vertex in the worst case due to skewed-tree formation, where V is the number of vertices in the graph. Here, we can see that time complexity for cycle detection has significantly improved compared to the previous approach.

Time Complexity of Kruskal's Algorithm:
In our code, we have the following three steps: (Here, the total number of vertices is n, and the total number of edges is E)

- Take input in the array of size E.
- Sort the input array based on edge-weight. This step has the time complexity of O(E log(E)).
- Pick (n-1) edges and put them in MST one-by-one. Also, before adding the edge to the MST, we checked for cycle detection for each edge. For cycle detection, in the worst-case time complexity of E edges will be O(E.n), as discussed earlier.

Hence, the total time complexity of Kruskal's algorithm becomes O(E log(E) + n.E). This time complexity is bad and needs to be improved.
We can't reduce the time taken for sorting, but the time taken for cycle detection can be improved using another algorithm named **Union by Rank and Size, Path Compression**. The basic idea in these algorithms is that we will be avoiding the formation of a skewed-tree structure, which reduces the time complexity for each vertex to **O(log(E))**.